國立臺灣大學工學院工業工程學研究所

碩士論文

Graduate Institute of Industrial Engineering

College of Engineering

National Taiwan University

Master Thesis

平行演算之優加劣減蟻拓優化法求解旅行銷售員問題

Parallel Superior/Inferior Segment-Discriminated Ant System for Travelling Salesman Problems

李湛

Chan, Lee

指導教授：楊烽正 博士

Advisor: Feng-Cheng Yang, Ph.D.

中華民國 102 年 7 月

July, 2013

國立臺灣大學碩士學位論文
口試委員會審定書

論文中文題目：平行演算之優加劣減蟻拓優化法求解旅行銷售員問題

論文英文題目：Parallel Superior/Inferior Segment-Discriminated Ant System for Travelling Salesman Problems

本論文係李 湛君（學號 R00546022）在國立臺灣大學工業工程學研究所完成之碩士學位論文，於民國 102 年 7 月 30 日承下列考試委員審查通過及口試及格，特此證明

口試委員：

楊烽正（指導教授）

洪一薰

蔡瑞煌

羅士哲

系主任、所長：陳正剛

致謝

用盡血淚拚搏，奮不顧身衝撞每一階段升學的窄門，所有的研究生來到了臨風顧盼的高度，成就是屬於任何直接或間接將資源投資在我們身上的你們，謝謝。

但我要首先對不起這個社會，我晚了，佔用了你們的資源只為了成就自己，我很怕這兩年來的風風雨雨，充其只是在實務界的庸庸碌碌，我抵達了碩士學位的好望角，但沒有任何的職場經驗，學術界的菁英在踏進職場新大陸後又是魯蛇重新學習自我塑造的開始，我期許我能將的成就能夠化為貢獻，得以回饋在真正被需要的地方。

謝謝媽、謝謝爸、謝謝妹、謝謝惠文*、謝謝楊老師、謝謝宏國（或國宏）、謝謝小黑、謝謝冠慶、(謝謝旅寧**)、謝謝佳璇、謝謝宗皓.....(以下省略百萬字)，對我而言，你們就是我世界的全部，你們的言行都是我自我激勵或省身的契機，未來還請多多賜教，thanks for all。

*女朋友

**他後來換指導老師了

李湛　2013 年 7 月

中文摘要

啟發式演算法為眾所皆知處理高複雜度問題的技術，但仍在現實中的高維度問題下的運算時間上仍有物理的計算瓶頸存在；而近二十多年來，多執行緒(Multi-Thread)的平行運算的程式開發技術儼然已經成了加速演算速度的重要趨勢。因此本研究使用 nVDIA 的 CUDA 平行運算開發平台搭配優加劣減蟻拓優化法(Superior/Inferior Segment-Discriminated Ant System,SDAS)來求解旅行銷售員問題(Travelling Salesman Problem)，並針對平行運算搭配 SDAS 下的不同演算策略進行進一步的研究分析。

關鍵字：CUDA、平行運算、蟻拓最佳化技術、優加劣減蟻拓優化法, 旅行銷售員問題

Abstract

The meta-heuristic algorithm, known as a technique coping with high complexity problems, has still been encountered the physical bottleneck calculating under the high dimension problems in realistic problems. Recent twenty years, moreover, the Multi-Thread parallel calculating program developing techniques has obviously became a vital trend accelerating the speed in calculation speed and effeciency of algorithm.

Therefore, the thesis approaches the Travelling Salesman Problem (TSP) , based on the CUDA, a parallel computing platform and programming model created by nVDIA, together with the Superior/Inferior Segment-Discriminated Ant System (SDAS). This thesis also looks for further research analysis in connection with different algorithm strategies in collection with the CUDA and the SDAS.

Keywords: CUDA, SDAS, TSP, parallelization

目錄

中文摘要.....i
Abstract.....ii
目錄.....iii
圖目錄.....v
表目錄.....vi
第一章 緒論.....1
1.1. 研究背景.....1
1.2. 研究動機與目的.....2
1.3. 研究範疇與架構.....3
1.4. 章節概要.....5
第二章 文獻探討.....6
2.1. TSP 問題.....6
2.2. 蟻拓最佳化技術.....7
2.3. 平行化演算技術應用於啟發式演算法.....11
2.4. 文獻探討結語.....14
第三章 旅行銷售員問題及其平行化 SDAS 蟻拓求解.....15
3.1. 旅行銷售員問題定義.....16
3.2. SDAS 蟻拓求解法.....16
3.2.1 較佳較差界限法.....17
3.2.2 優加劣減費洛蒙矩陣更新法則.....19
3.3. CUDA 及平行化設計概念.....21
3.3.1. GPU 硬體環境.....22
3.3.2. GPU Kernel 記憶體管理及設計概念.....22

3.4. SDAS 求解 TSP 的序列化演算程序設計 24
3.5. SDAS 用於 TSP 平行化設計－途程解建構 26
3.5.1. 途程解建構 kernel－設計概念.....27
3.5.2. 途程解建構 kernel－程式設計.....30
3.6. SDAS 用於 TSP 平行化設計－更新費洛蒙矩陣 36
3.6.1. 更新費洛蒙矩陣 kernel.....36
3.7. 小結.....40
第四章 平行化 SDAS 及範例驗證 41
4.1. 平行化 SDAS 求解系統實作 41
4.2. 演算法成效分析.....42
4.2.1. 平行化 SDAS 與 SDAS 之同質性比較.....43
4.2.2. 各演算階段的加速結果比較.....45
4.2.3. 整體平行化加速結果比較.....55
4.2.4. atomic 及 partition 兩種資料存取模式的結果比較.....58
4.3. 小結.....60
第五章 結論與未來研究建議.....62
5.1. 結論.....62
5.2. 未來研究建議.....63
參考文獻.....64
附錄.....66

圖目錄

圖 1.1 研究流程圖.....4

圖 2.1 螞蟻覓食示意.....8

圖 2.2 輪盤法比例分配示意.....10

圖 2.3 Master & slave 平行演算示意.....12

圖 2.4 平行化蟻拓演算法的類型示意樹狀圖.....13

圖 2.1 優段劣段集合示意.....21

圖 3.2 block 大小為 7，thread 大小為 8 的執行緒區塊示意圖..23

圖 3.3 記憶體與 device 間的關聯關係示意圖.....24

圖 3.4 城市連結標誌矩陣 **T** 範例.....27

圖 3.5 城市連結標誌矩陣 $\mathbf{T}_m$ 範例.....29

圖 3.6 Reduction max 示意.....32

圖 3.7 Data-based 平行化演算示意.....33

圖 4.1 建構途程解 kernel 的平行程式運算時間比較.....47

圖 4.2 建構途程解 kernel 的平行程式 speed up 比較.....47

圖 4.3 更新費洛蒙矩陣 kernel 的演算時間比較.....50

圖 4.4 更新費洛蒙矩陣 kernel 的 speed up 比較.....50

圖 4.5 SDAS atomic 模式各階段演算時間比例.....53

圖 4.6 SDAS partition 模式各階段演算時間比例.....53

圖 4.7 平行化 SDAS 不同模式下整體演算時間比較.....54

圖 4.8 平行化 SDAS 與 MMAS 的 speed up 比較.....57

圖 4.9 平行化 SDAS 與 MMAS 的演算時間比較.....57

圖 4.10 求解 rat783 下變動求解代次上限的 speed up 比較.....59

圖 4.11 求解 rat783 下變動螞蟻數的 speed up 比較.....60

表目錄

表 4.1. 範例測試執行求解系統的硬體環境.....42

表 4.2. 序列化 SDAS 與平行化 SDAS 於不同例題下的比較.....44

表 4.3. 序列化 SDAS 與平行化 SDAS 於不同例題演算的平均執行時間比較.....44

表 4.4. 建構途程解 kernel 加速比較.....46

表 4.5. 更新費洛蒙矩陣 kernel 演算耗費時間及加速比.....49

表 4.6. 兩主要演算模式運算耗時比例.....52

表 4.7. 演算法整體運算耗時及加速成效比較.....56

表 4.8. 在不同螞蟻數與不同求解代次上限下求解 rat783 的運算時間和加速結果比較.....59

第一章 緒論

在使用程式輔助啟發式演算法求解下，已經遭遇了硬體設備的瓶頸。隨著問題的維度增加，如果可以運用 GPU 的多執行緒的程式開發技術來降低計算所花費的時間，甚至是解的品質，都是演算機制上的重要里程碑。

1.1. 研究背景

旅行銷售員問題(Travelling Salesman Problem, TSP)屬於 NP-Hard 問題(Dantzig 等人, 1959)。為最具代表性的離散優化問題，旨在規劃一旅行銷售員由起點出發，行經所有給定需求點後，最後再回到原點的最小路徑成本。對應到現實問題中的物流配送公司，如何在將所有客戶的訂單全部送到的情況下，確定最短路線。在最佳化這樣高度複雜的優化問題時，啟發式演算法的研究則日益重要。

「蟻拓最佳化技術(Ant Colony Optimization, ACO)」在啟發式演算法的發展中有著不可磨滅的地位，其為類蟻拓尋優演算步驟方法的統稱，透過觀察自然界中螞蟻的覓食、合作精神，模仿其利用費洛蒙彼此溝通信息以最佳化食物搬運的路徑，所設計逐步建構可行解的最佳化技術。

nVIDIA 推出了 CUDA 的平行運算程式開發平台，可透過相對較低的成本進行高速運算，啟發式演算法加速的研究自此開始逐漸在學界加溫，尤以區域搜尋法為甚。

1.2. 研究動機與目的

一般啟發式演算法所需的演算時間會隨著問題的維度而成指數成長，儘管有再好的求解品質，也意味著所需的時間成本也相應的增加；因此也有許多研究致力於加快求解的速度，同時亦不會劣化解的品質。

優加劣減蟻拓優化法(Yang & Chou, 2009) (Superior/Inferior Segment-Discriminated Ant System, SDAS) 在 蟻拓演算法中以解高複雜度的組合最佳化及分群優化問題聞名。但在求解高維度問題時仍然有一般啟發式演算法硬體速度上的瓶頸，因此本研究希望能應用平行運算的程式開發技術優化 SDAS 來求解。

平行化運算技術 (Parallelization Strategies) 是近年在高效能運算上的顯學，但一直以來由於其在程式撰寫上的門檻相對不易跨越，也導致本來就不易平行化的啟發式演算法搭配平行化運算技術的研究成果裹足不前。不過自從 NVidia 開始在繪圖晶片中，加入了用來支援 CPU 平行運算的 CUDA 技術後，技術支援的機動性大增，程式撰寫的門檻逐年下降，且兼具穩定性及優越的運算效能，在除錯機制上也有不錯的表現。

本研究旨在搭配平行運算技術及 SDAS 求解 TSP 問題並探討不同演算策略下的求解品質及效能。

1.3. 研究範疇與架構

本研究使用平行運算技術來搭配 SDAS 求解 TSP 問題，在 CUDA 程式撰寫平台上實作原序列化的 SDAS 並與此後將之平行化對證改良成果。本研究的研究流程圖如圖 1.1 所示。

本研究研讀平行運算撰寫技巧及利用平行運算技術改良啟發式演算法的相關文獻，萌生研發動機及目的。確立研究方向後，著手序列化 SDAS 並緊接將之平行化，同時不忘研讀相關文獻尋求任何改良的可能性，並持續改進演算法及增添系統各項功能。此程序反覆進行，並且確認程式功能是否完備沒有錯誤，最後與相近之文獻的方法比較、分析和討論。

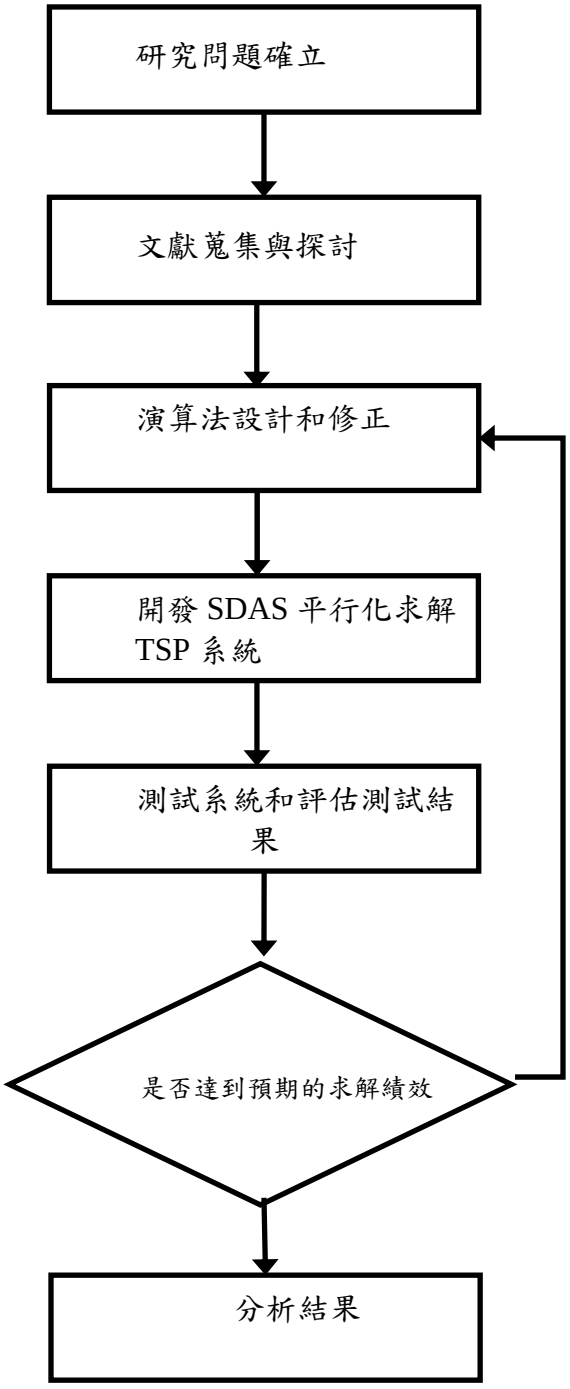

研究問題確立
文獻蒐集與探討
演算法設計和修正
開發 SDAS 平行化求解 TSP 系統
測試系統和評估測試結果
是否達到預期的求解績效
分析結果

圖 1.1　研究流程圖

1.4. 章節概要

本論文主要共分為五個章節，編排方式如下：第一章為緒論，概述本研究背景、動機與目的。第二章為文獻探討，針對利用 GPU 平行演算優化蟻拓演算法前學者的相關研究，TSP 的定義與蟻拓優化技術的運作發展做整理。第三章詳述本研究提示的求解模式以數學模型呈現，並針對 SDAS 的解建構途程與更新費洛蒙值做 CUDA 的平行化設計，並以範例介紹平行化的概念與具體作法。第四章呈現使用本研究提出的平行化演算法測試標竿問題的結果，並就結果分析討論。第五章歸納總結出未來可進行的研究方向，供有興趣的學者做參考。最後附錄的部分列出本研究平行化 SDAS 求解 TSP 問題的 CUDA 程式碼。

第二章 文獻探討

本章節首先介紹旅行銷售員問題（Travelling Salesman Problem, TSP），接著回顧蟻拓最佳化技術的起源，進而探討本研究所著重的 SDAS，接著帶出平行運算應用在啟發式演算法上的相關文獻，及相關學術定義以及分類指標，最後引導出與蟻拓演算法及平行運算技術搭配的相關技術。本章可分為以下五節，第一節介紹本研究開發平行化程式時所拿來展示的 TSP 問題；第二節探討蟻拓最佳化技術；第三節概述截至目前平行化演算技術應用於啟發式演算法的發展與研究；第四節文獻回顧小結。

2.1. TSP 問題

在組合最佳化的問題中，旅行推銷員問題（Traveling Salesman Problem）最具有代表性，該問題只消經過數學上適當的轉換，便得以運用在許多實務的問題上。如：車輛途程問題、定向越野問題、印刷電路板元件插製程問題等均可模式化為旅行推銷員問題。以下將介紹旅行推銷員問題的數學模型定義。

若以圖論（Graph theory）定義旅行推銷員問題，可用網圖 $\mathbf{G}$ 代表城市間的路網結構，$\mathbf{G}=(\kappa,\mathbf{L},\mathbf{C})$。其中，$\kappa$ 是路網的節點集合，TSP 中是所有程式編號的集合 $\kappa=\{1,2,...,z\}$；而 $\mathbf{L}$ 程式間連結關係的集合，代表所有程式間連結路段的集合。城市 i 至城市 j 的路段連結 $l_{ij}\in\mathbf{L}$。其長度 c_{ij} 代表城市 i 與城市 j 間的距離。典型的旅行推銷員問題，是推銷員可從任一城市出發，在滿足經過路網中所有城市一次，並回到出發城市的條件，求得一條總長度最短的路線。該路線即是圖論中所提及的漢彌爾頓迴路（Hamiltonian Circuit）。

2.2. 蟻拓最佳化技術

蟻拓最佳化技術（Ant Colony Optimization, ACO）在眾多啟發式演算法中身世顯赫，其所代表的是具有類似蟻拓最佳化演算步驟的統稱。蟻拓最佳化技術起源於模仿真實螞蟻的覓食行為，螞蟻在進行群體通訊時的行動主要依靠嗅覺及排放費洛蒙（Pheromone）。費洛蒙會遺留在螞蟻所走過的路徑上，用來傳遞路徑選擇訊息給其他螞蟻。較多螞蟻行經的路徑會因為重複遺留費洛蒙而留下較多的費洛蒙氣味，而較少走過的路徑則會因為費洛蒙蒸發而逐漸變弱。但並非所有的螞蟻都會完全的依循費洛蒙的導引前進，偶爾有部分的螞蟻會脫離主要幹道，嘗試另闢蹊徑，而無論如何，最終將會有一條路徑上的費洛蒙強度額外顯著，繼而所有螞蟻都會因循同樣的路徑行進，此路徑往往是從巢穴到食物間的直線且接近最短路線。

圖 2.1 描述上述所提的覓食過程，當螞蟻在覓食途徑中遇到障礙物時，透過費洛蒙來揀選較短路徑的過程。假設有一蟻群正在食物和巢穴間來回往返，如圖 2.1 (a) 所示。頃刻間，此路徑上多了一層障礙物，因而中斷了螞蟻原本的行走路徑，如圖 2.1 (d) 所示但螞蟻最終還是會在岔路上選擇較短的路徑行走。

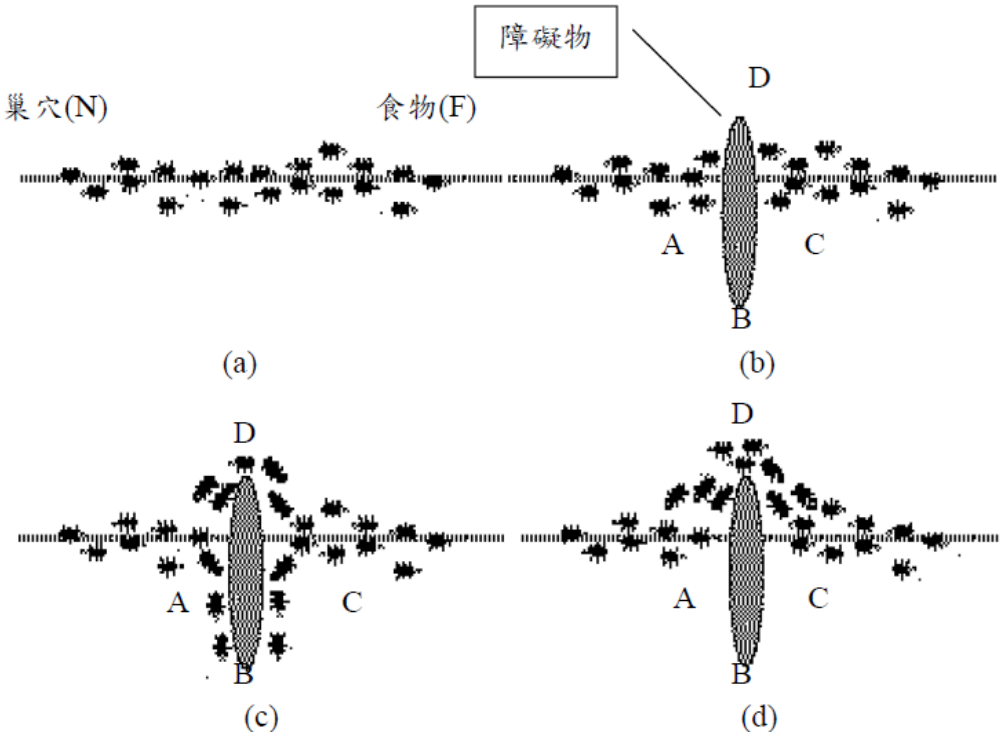


圖 2.1　　螞蟻覓食示意

圖中的情境分述如下：

(a) 假設一開始巢穴和食物間並無任何障礙物，所以螞蟻會隨著費洛蒙的氣味在巢穴和食物間往返。

(b) 加入了一個障礙物後，對螞蟻而言路徑上產生了岔路，此時，螞蟻選擇兩邊 (路徑 ADC 和路徑 ABC)的機率是相同的。

(c) 路徑 ADC 和路徑 ABC 上都有螞蟻在上面行走，但由於螞蟻行走的速度相當，所以若路徑越長則費洛蒙會容易因為蒸發而較不能夠留存。

(d) 最後路徑 ADC 上累積較多的費洛蒙量，而路徑 ABC 上的費洛蒙因蒸發殆盡而幾乎不會再有螞蟻選擇行走。

從上述真實螞蟻的覓食行為，可以歸納出螞蟻覓食有下列幾點特色：(1)利用團體合作的方式來完成工作。(2)透過費洛蒙來達成相互間的溝通。(3)最後會達到共識。這 3 點特色，是把真實螞蟻的行為模式化成蟻拓尋優法的重要依據。

螞蟻系統

螞蟻系統（Ant System, AS）(Dorigo, Maniezzo, & Colorni, 1996)為螞蟻最佳化技術最早的版本。最先被應用於旅行推銷員問題（Traveling Salesman Problem, TSP）中，之後其他學者陸續將此演算法運用於其他問題。例如零工式生產問題、二次指派問題等。所有蟻拓最佳化技術都是以 AS 作為基本架構，因此以下將以 AS 求解旅行推銷員問題為例，說明演算流程和求解架構。

給定n 位顧客，i 是顧客的索引編號，TSP 問題目標是找到一個最短的封閉迴路，使得推銷員能恰巧拜訪每一位顧客一次。每隻螞蟻可視為擁有記憶功能的代理人（Agents）。他的記憶包含禁忌名單（Tabu List）和候選清單(Candidate List) Z 。禁忌名單紀錄已拜訪過的顧客，用來確保螞蟻在前進每一步（Step）時不會選到重覆的顧客。而候選清單則是記錄螞蟻下一步可以選擇的顧客集合。換句話說，紀錄在候選清單內的顧客，才有可能在螞蟻下一步時被選擇。

假設η_{ij} 是關連啟發式運算值，在求解 TSP 時常指顧客i 到顧客j 間的距離的倒數，又被稱作能見度（Visibility）。τ_{ij} 是顧客i 到顧客j 間的費洛蒙強度，即顧客i 與顧客j 間的關連強度，而α 和β 分別是費洛蒙強度和關聯啟發式運算值的權重參數，讓使用者自行設定。事實上，α 和β 可以說是「眼前最好」與「經驗最好」間的取捨和平衡。因此，AS 設計螞蟻從顧客i 到顧客j 間的轉移機率（Transition Probability）。

$$P_{ij}=\begin{cases}\dfrac{\tau_{ij}^{\alpha}\cdot\eta_{ij}^{\beta}}{\sum_{g\in Z}\tau_{ig}^{\alpha}\cdot\eta_{ig}^{\beta}} & \text{, if } j\in Z\\ 0 & \text{, otherwise}\end{cases}。 \tag{2.1}$$

需注意的是，若螞蟻是選擇最大轉移機率值來決定下一個加入的物件時，會容易使求解陷入區域最佳解。為了避免上述情形，螞蟻採用輪盤法（Roulette Wheel Selection）的比例隨機模式（Proportional Randomness）來選擇下一個要加入的物件。如圖所示，假設目前候選清單內有五個候選顧客，且分別的轉移機率值為 $P_1=0.3$、$P_2=0.15$、$P_3=0.2$、$P_4=0.25$、$P_5=0.1$。使用輪盤法需把機率值轉換成機率區間，因此每個顧客的機率區間　分別為：$P_1'=[0,0.3)$、$P_2'=[0.3,0.45)$、P_i'

$P_3'=[0.45,0.65)$、$P_4'=[0.65,0.9)$、$P_5'=[0.9,1.0]$。當螞蟻要從這 5 個顧客中選出一個時，先隨機產生一範圍在 0.0 與 1.0 間的亂數，此亂數著落的數值區間所映對的物件即是中選顧客。

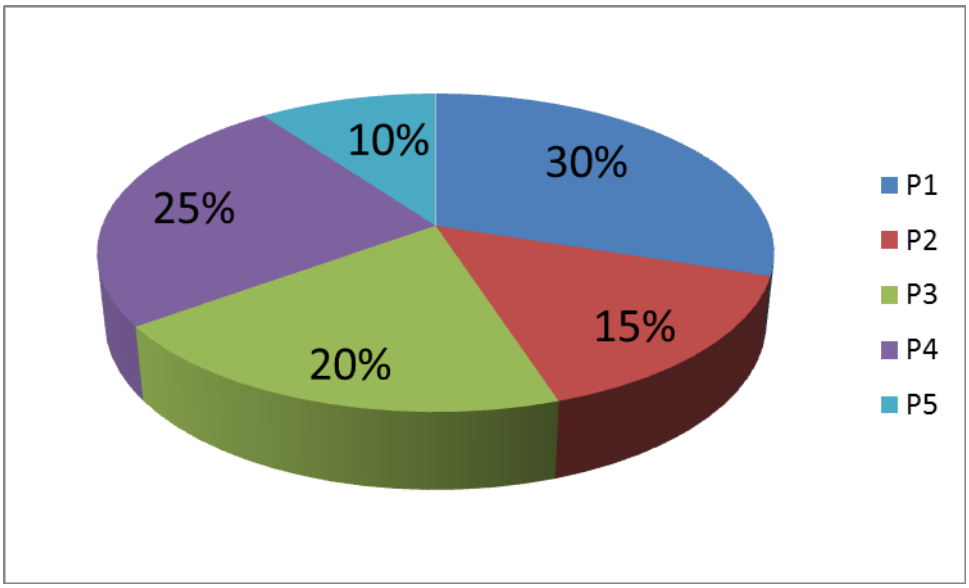


圖 2.2　輪盤法比例分配示意

例如亂數值為 0.42 時，落在 P_2' 區間，因此選擇顧客 2 為後續服務的對象。由此可發現輪盤選擇法是採用比例式的隨機方式進行物件選取，因此佔有輪盤上愈大面積的物件愈有機會被選中。此法加入隨機的因子，可讓螞蟻在建構解過程較不容易陷入區域最佳解，且探索較大的解空間有較高的機會搜尋到全域最佳解。

當全部的螞蟻都各自建構出一組解後，會使用費洛蒙更新法則

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \Delta\tau_{ij}, \forall i, j$$

$$\Delta\tau_{ij} = \begin{cases} \dfrac{Q}{L} & \text{,if the ant travel from } i \text{ to } j \\ 0 & \text{,otherwise} \end{cases} \tag{2.2}$$

此法則用來更新兩顧客間的費洛蒙強度，其中參數 Q 是一個常數，其值會依問題結構不一，由使用者自行設定。而 L 是螞蟻所完成的路徑總長度。$\Delta\tau_{ij}$ 是顧客 i 和顧客 j 間的費洛蒙添加量。ρ 是費洛蒙蒸發率 (Evaporation Rate) ，換言之 $(1-\rho)$ 則是費洛蒙經蒸發留下的量。其值的設定會影響螞蟻趨向區域最佳解的速度，越大會造成求解的收斂速度越快，但求解品質會降低。因此該值的設定完全取決於使用者對求解時間及求解品質間的權衡。

2.3. 平行化演算技術應用於啟發式演算法

在平行化啟發式演算法的技術中，沿用 Crainic 等學者的分類(Crainic & Toulouse, 2003)，可將平行演算策略分成三種類型：

1.演算法的資料結構與演算程序不變，不同執行緒間彼此獨立互不干涉，單純增加求解代理人的數量加速演算時間。

2.將演算法分解成 master 及 slave 兩種不同的決策層級，如圖 2.3 所示(The, x, Van, Melab, & Talbi, 2013)，解代理人解建構完成後，便以此解（master）為依據，對鄰近解做區域搜尋（slave），相對於第一種類型，master 及 slave 的求解方式在平行化演算的研究中展有最大的多數，此方法也稱作為（Local search meta-heuristics, LSMs)。圖中解 $\{4,3,4\}$ 為 master，下面虛線內的解為變動 master 內其中一個解順位的鄰近解，皆為 slave。

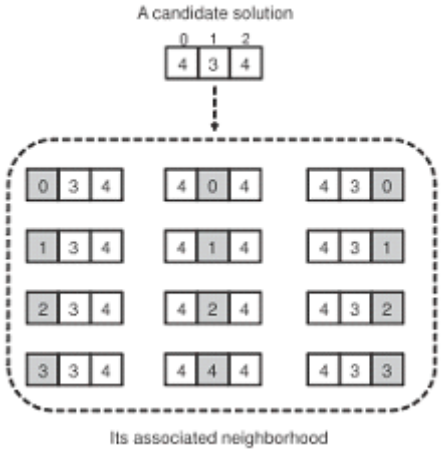


圖 2.3　Master & slave 平行演算示意

3.重新將演算法改寫，使演算的資料結構在相對於序列化的演算流程中具有優勢，近年內的研究趨勢都向此類型發展，在配合硬體效能與開環境下設計的平行化演算程式，往往都有出眾的表現。

本研究旨藉由 CUDA 程式撰寫平台設計將原序列 SDAS 演算平行化，其中 Pedemonte 等人(Pedemonte, Nesmachnow*, & Cancela, 2011)所發表對 2010 年前對針對平行化蟻拓演算法的統整給了本研究摸索時期可觀的指引。Pedemonte 等人將蟻拓演算法的平行化區分成六種類型，如圖 2.4 所示

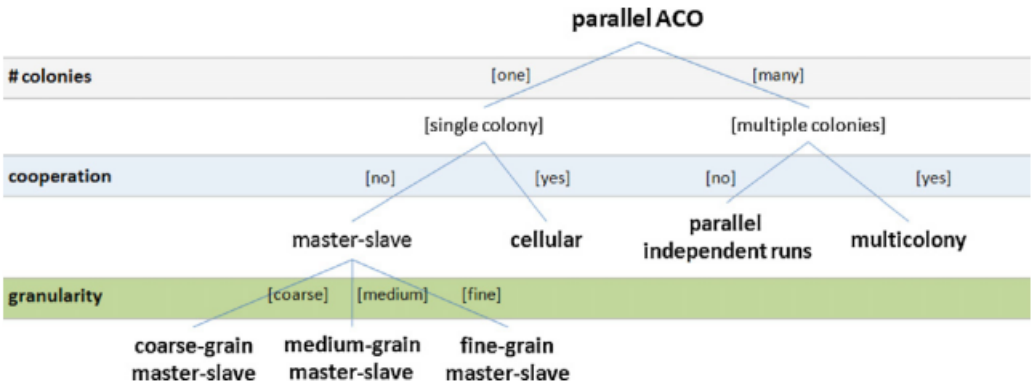


圖 2.4　平行化蟻拓演算法的類型示意樹狀圖

coarse-grain master-slave：屬於 master-slave 類型，master 的任務是費洛蒙矩陣的更新，slave 的任務是途程解建構。應用在 CUDA 平行化演算的問題有定向越野問題(Catala, Jaen, & Modioli, 2007) 、及連續優化問題(Weihang & Curry, 2009)。

medium-grain master-slave：屬於 master-slave 類型，master 在途程解建構到一半後便交付 slave 將剩下的解當作 local search 進行演算。

fine-grain master-slave：屬於 master-slave 類型，slave 僅進行極少的資訊交換，主要演算皆由 master 進行。曾經有被應用於 MMAS 求解 TSP 問題(Wang, Dong, & Zhang, 2009)

parallel independent runs：每隻螞蟻皆在各自的費洛蒙矩陣做更新，螞蟻間不會經常交換費洛蒙資訊。曾經被應用於 MMAS 求解 TSP 問題(Hongtao, Dantong, Ximing, Lili, & HaiHong, 2009)

cellular 及 multicolony 截至目前為止的研究皆使用 cluster 進行平行運算，不在本研究關注的範疇內因此不做深入討論。

2.4. 文獻探討結語

近年來，平行化演算技術因為 CUDA 的出世，而有了有別於傳統單執行緒 (Single-Thread) 的程式設計思維，多執行緒 (Multi-thread) 程式架構在執行演算時著重於執行緒間的相互合作與演算資源分配，在蟻拓演算法的演算流程已被摸透的今日，著實提供了與眾不同的研究向度，本研究嘗試應用 CUDA 平行化程式開發平台利用 GPU 加速優加劣減螞蟻系統應用於 TSP 問題。在下一章將詳細說明優加劣減螞蟻系統的演算求解程序。

第三章 旅行銷售員問題及其平行化 SDAS 蟻拓求解

使用通用圖形處理器（General-purpose computing on graphics processing units, GPGPU） 執行啟發式演算法，相較於典型的單執行緒（Single-Thread）啟發式演算程式，具有高速求解速度的優勢。因此面對高維度，高複雜度的問題求解時應能展現高求解效能。近年來由於 CUDA 的平行運算程式開發平台日益發達，GPU 配備功能提高價格日益便宜，有越來越多的啟發式演算法開始嘗試進行演算平行化後，在 CUDA 架構下執行，另也因為硬體規格上的相異，再加上 GPU 特殊的記憶體儲存機能，演算程序設計需要考量的因素不侷限於以往 CPU 堆疊平行化的觀念，其中記憶體的分配、傳送甚至會隨著配置的記憶體型態不同，而有不同的處理速度。GPU 的運算功能在繪圖運算，演變至今逐步被應用於各式各樣的高速平行運算，因其特殊的硬體結構，衍生出多元的應用及研究方向，也存在著很大的研究空間。

本研究旨在研擬優加劣減蟻拓優化法（SDAS）的平行化演算模式，求解旅行銷售員問題（Travelling Salesman Problem, TSP），研究內容除了實作求解系統外並進行範例測試以驗證平行化的成效。本章首先介紹 TSP 問題及 SDAS，再說明平行化的概念，並研擬 SDAS 平行化的程序，過程會以小規模的範例解說並使用 psudo-code 說明演化程序，最後再研擬平行化的區域搜尋技法增加演算法的求解效能。

3.1. 旅行銷售員問題定義

本研究將藉由旅行銷售員問題來研擬及驗證 SDAS 的平行演算成效，旅行銷售員問題中的推銷員可從任一城市出發，繞經所有城市一次，並回到出發的城市，問題的目的在於求得一條最短行經路徑，本節簡述旅行銷售員問題及其基本假設，再定義數學模式。

Hassler Whitney 於 1934 年提出旅行銷售員問題，是一代表性的組合最佳化的 NP-complete 問題，問題求解概念簡單但求解難度大。問題的求解目標在最小化旅行總路徑長。旅行銷售員須恰好行經所有城市一次，城市總數為 n，而且路徑將形成一條路徑最短的封閉迴路，歸類為 NP-hard 組合最佳化問題。在歐幾里德模式下的 TSP (Euclidean TSP) 中，城市 i, j 間的距離是以歐式距離計算，設距離 $c_{ij}=\sqrt{\left(x_i-x_j\right)^2+\left(y_i-y_j\right)^2}$，其中 $\left(x_i,y_i\right),\left(x_j,y_j\right)$ 分別代表城市 i, j 的 X 與 Y 軸的座標，城市編號的集合為 $\delta=\{1,2,...,n\}$，而 L 是城市連結關係的集合。

一隻螞蟻所建構的解即為城市拜訪順序 $S=\langle s_1,s_2,...,s_n\rangle, s_i\in\delta, s_i\neq s_j$，符號 s_i 代表第 i 座所行經的城市編號，可行解 S 的目標函數值即為總行經距離

$$f(\mathbf{S})=\sum_{i=1}^{n-1}c_{S_iS_{(i+1)}}+c_{S_nS_1} \tag{3,1}$$

3.2. SDAS 蟻拓求解法

螞蟻在大自然中演化至今，進化出了一種以集體行動為方針的覓食方式，極小化覓食路徑以確保行進效率以及盡量避免行經危險區域。每隻螞蟻離開巢穴後便各奔東西尋找食物，沒有視覺的螞蟻，仰賴的是螞蟻經年演化後所產生的獨特分泌物費洛蒙（pheromone），費洛蒙會在螞蟻行進時分泌在所行經的路徑上，

作為行進記號，找到食物的螞蟻會與其他螞蟻透過觸角溝通，同時分泌更多費洛蒙標示食物途程在回到巢穴之中，其餘螞蟻在離開巢穴探索時，若選擇跟隨此路徑且找到了食物來源，便會依循同樣的機制分泌費洛蒙，這樣一來該路徑上的費洛蒙濃度將逐漸變強，沒有螞蟻跟隨的路徑上的費洛蒙將逐漸揮發減弱。當路徑上的費洛蒙濃度越高，此路徑將會吸引更多的螞蟻經過，最後則會形成一條從巢穴到食物的主要路徑，且往往是最短路徑。簡而言之，螞蟻路徑選擇的行為主要取決於費洛蒙濃度，因此費洛蒙是螞蟻之間的主要溝通管道。

而 SDAS(林典翰, 2004)即是啟發自螞蟻總是以最短路徑覓食自然現象的演算法，承襲自螞蟻系統（Ant System, AS）啟發式演算法，透過模仿螞蟻覓食的行為，再螞蟻解建構的過程中，每移動一步就會執行逐步更新費洛蒙機制，當派出的每一隻螞蟻都完成解的建構後，進行費洛能的更新，將螞蟻依照解的品質分為優蟻、劣蟻和普通的螞蟻，採用不同的入圍策略來決定餐與費洛蒙更新的螞蟻。在 SDAS 中首先採用「較佳較差界限法」，此法依據統計品管中 $\bar{x}-R$ 管制圖的概念定義出較佳目標函數值上限和較差目標函數值下限，以目標函數值的表現將螞蟻區分成較佳、較差、及一般三群後，接著再以「隨機差及比對」定義出優段及劣段城市連結集合後，在分別強化優段及弱化劣段上的費洛蒙值。

3.2.1 較佳較差界限法

在更新費洛蒙矩陣時必須權衡資訊的充分利用或資訊的有效利用兩者，若所有螞蟻皆可回傳資訊，則可能會限制演算法的求解品質，相對地如果只有菁英螞蟻能夠回傳資訊，又失去了蟻拓演算法螞蟻間合作的精神，因此為了充分將兩者優點結合，「較佳較差界限法」因應而生，首先 m 隻螞蟻求算其目標平均目標函式值

$$\bar{f}=\frac{1}{m}\sum_{k=1}^{m} f^{k}(\mathbf{S})\ , \qquad (3,2)$$

利用 $\bar{x}-R$ 管制圖的概念定義出較佳目標函數值上限

$$f_{\max}=\max\left\{f^{k}(\mathbf{S})\,|\,\mathrm{k}=1,2,...,\mathrm{m}\right\} \tag{3,3}$$

和較差目標函數值下限

$$f_{\min}=\min\left\{f^{k}(\mathbf{S})\,|\,\mathrm{k}=1,2,...,\mathrm{m}\right\}\text{，} \tag{3,4}$$

以極小化問題為例較佳界限則為

$$F_{best}=\bar{f}-\varpi\left(\bar{f}-f_{\min}\right)\text{，} \tag{3,5}$$

同時較差界限為

$$F_{worst}=\bar{f}+\varpi\left(f_{\max}-\bar{f}\right)\text{。} \tag{3,6}$$

其中參數 ϖ 用以控制管制界限的幅度，其值介於 0 到 1 之間。當 ϖ 越接近 1 則代表螞蟻所帶的解其解函數值需要越接近目標函式上限或下限才能夠被選入為優蟻或劣蟻。

而在極小化問題中，可以得到較佳螞蟻群的編號集合

$$B=\left\{k\,|\,f(\mathbf{S})^{k}\le F_{best},\text{where } k\in\{1,2,...,m\}\right\} \tag{3,7}$$

以及較差螞蟻群的編號集合

$$W=\left\{k\,|\,f(\mathbf{S})^{k}\ge F_{worst},\text{where } k\in\{1,2,...,m\}\right\}\text{。} \tag{3,8}$$

由此一來較佳蟻群與較差蟻群可分別參看其解的城市連結對費洛蒙矩陣的更新行使不同的策略，有助於區分出需要添加費洛蒙的城市連結及需要淘汰的城市連結資訊。

3.2.2 優加劣減費洛蒙矩陣更新法則

在區分出那些螞蟻入圍餐與調整費洛蒙矩陣的工作後，為了將螞蟻中的解連結資訊取出，將執行拆解運算式 $\mathbf{H}^{pheromone\ type}(\mathbf{S})$ 求得該解的城市連結集合，如 $\mathbf{S}=\{1,4,6,3,2,7,5\}$，則 $\mathbf{H}^{pheromone\ type}(\mathbf{S})=\{(1,4),(4,6),(6,3),(3,2),(2,7),(7,5),(5,1)\}$，同時也表示成 $\mathbf{H}^{pheromone\ type}(\mathbf{S})=\{l_{12},l_{23},l_{34},l_{45},l_{56},l_{67},l_{71}\}$，依照 3.2.1 節中所區分之優蟻劣蟻以及迄今最佳解將城市連結分為三個集合，迄今最佳解城市連結集合

$$\mathbf{G}=\mathbf{H}(\mathbf{S}^{*})\ , \tag{3,9}$$

由式(3,7)結果決定優段連結集合

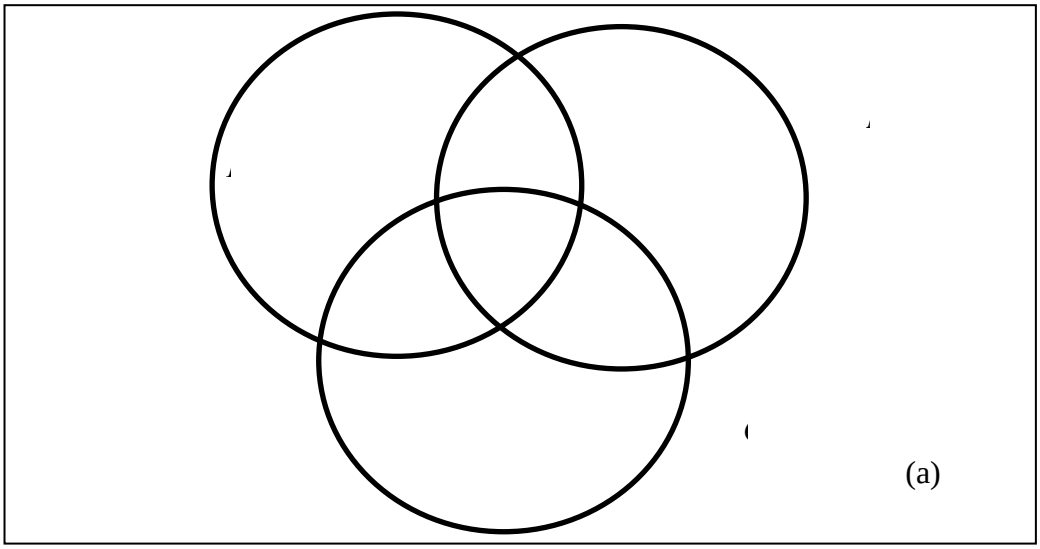
$$\mathbf{A}''=\underset{k\in\mathbf{B}}{\cup}\mathbf{H}(\mathbf{S}^{k}) \tag{3,10}$$

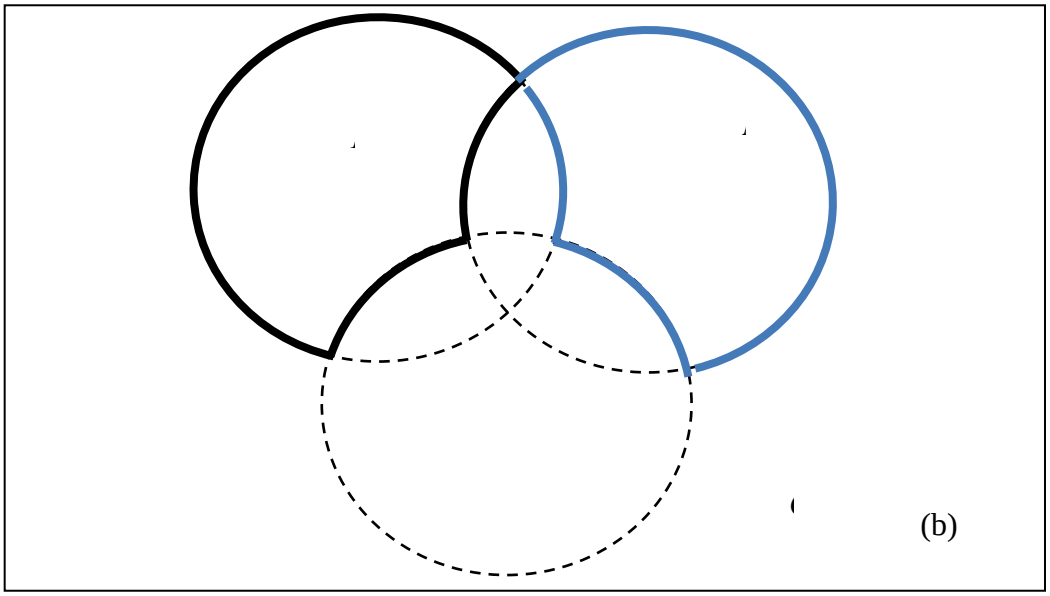
以及由式(3,8)結果決定劣段城市連結集合

$$\mathbf{D}''=\underset{k\in\mathbf{W}}{\cup}\mathbf{H}(\mathbf{S}^{k}) \tag{3,11}$$

其中 $\mathbf{G}$ 、$\mathbf{A}''$ 及 $\mathbf{D}''$ 三個城市集合間的關係可用文氏圖（Venn diagram）所表達，如圖 3.1(a)所示，同時為了將最佳解城市連結獨立於所有城市連結做更新費洛蒙矩陣的決策，因此再重新定義優段城市連結集合 $\mathbf{A}'=\mathbf{A}''-\mathbf{G}-\mathbf{D}''$ 和劣段城市連結集合 $\mathbf{D}'=\mathbf{D}''-\mathbf{G}-\mathbf{A}''$ 如圖 3.1(b)所示。為了配合蟻拓最佳化技術的求解精神，將優斷劣段城市連結集合進一步用隨機模式篩選，決定最後入圍參與決策的連結。優段城市連結定義為 $\mathbf{A}=\{l_{ij}\,|\,l_{ij}\in\mathbf{A}',\theta_{ij}<\theta^{A}\}$，劣段城市連結定義為 $\mathbf{D}=\{l_{ij}\,|\,l_{ij}\in\mathbf{D}',\theta_{ij}<\theta^{B}\}$，$\theta^{A}$ 為自訂優段門檻，θ^{B} 為自訂劣段門檻，當城市連結

l_{ij} 入圍 $\mathbf{D}'$ 或 $\mathbf{A}'$ 時，最後須進一步以隨機方式篩選，每個 l_{ij} 皆產生皆於 0 與 1 間的隨機變數 θ_{ij} ，當入圍優段的 l_{ij} 所對應的 θ_{ij} 小於設定的優段門檻 θ^A 時，l_{ij} 才能被歸為優段的城市，相對地當入圍劣段的 l_{ij} 所對應的 θ_{ij} 小於設定的劣段門檻 θ^B 時，l_{ij} 才能被歸為劣段的城市如圖 3.1(c)所示。





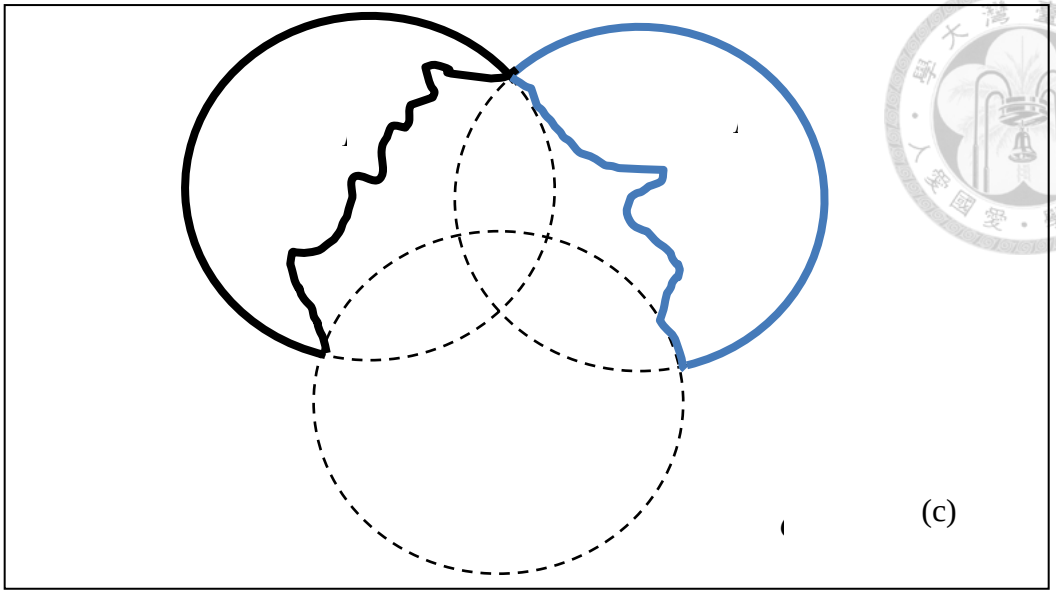


圖 3.1　優段劣段集合示意

SDAS 最主要的優勢為充分利用城市連結來對費洛蒙矩陣更新，隨後章節將利用本章提到的策略完整重現求解過程。

3.3. CUDA 及平行化設計概念

NVDIA 提出的 CUDA (Compute Unified Device Architecture) GPU 平行計算架構讓程式撰寫者可以藉由 CUDA 擴充的 C 語言在 GPU 上進行平行運算。可進行較以往早期 GPGPU 時代更輕鬆的方式撰寫程式，透過記憶體配置的規則及資料的分配，CUDA 架構下的 CPU 會將程式流程以及運算工作分配給多個 GPU 的執行緒 (threads) 執行。GPU 的 thread 則由硬體的計算核心 (cores) 執行。GPU 運算也是平價的高效能平行運算，能在低成本下得到優異的運算效能。GPU 平行演算多用於將大型問題拆解成多個相同且彼此不相關的小問題大量平行化後達到加速的效果。下面的各節分別說明 GPU 的硬體環境以及平行化演算時便用到的基本觀念。

3.3.1. GPU 硬體環境

GPU 最主要的構成是 Streaming Multiprocessors（ SMs ）以及專供傳遞給 GPU 運算的全域資料儲存記憶體（global memory）。SM 裡有存取 GPU 的主要運算元件 Streaming Processors（ SPs ）、以及可供每個 SP 各自獨立儲存資料的（share memory）共用記憶體。每個 SP 內有數個計算核心（core），可同時進行相同於 core 數量的執行緒平行運算。以 nVIDIA Quadro 600 GPU 卡為例，內有兩個 SP 組成單 GPU 的 Multiprocessors (MPs)。每個 SP 內有 96 個 core；Share memory 配有 64KiB；Global memory 配有 1023.69MiB。

3.3.2. GPU Kernel 記憶體管理及設計概念

在 CUDA 架構下程式的 GPU 運算環境是和 CPU 分開的。由 GPU 執行副程式的、函式、或方法，須由 CPU 環境下的主程式呼叫及傳入參數。CPU 稱為程式執行的主端（host），而 GPU 稱為設備端（device）。計算過程需要的資料及參數得從 host 傳遞至 device 上的記憶體，才能在 device 上進一步演算。在 device 執行的副程式，稱為 kernel 程式。CUDA 環境下的編譯器會將 kernel 程式編譯成 GPU device 能執行的機器碼。而 kernel 程式執行時可以同步啟動數百數千個 thread 進行平行運算。

執行 kernel 時，須設定執行緒區塊（ thread block ）的開設數量以及每個 block 中啟用的 thread 數以組成一個 grid 進行批次處理。不同的 GPU 卡硬體設備有不同的 block 數量設定上限及 thread 數設定上限。以 nVIDIA Quadro 600 GPU 為例，block 上限是 65535 個，而每個 block 中 thread 的上限是 1024 個。一般而言，呼叫 kernel 時的語法是 kernel_function<<<block-number, thread-number>>>(…) 在三層角括號內設定運算時所需要的 block 及 thread 數量。程式執行時 device 會依設定的 block 及 thread 數啟動 block 及 thread 並賦予不同的編號（ blockID &

threadID ）。kernel 程式則藉由執行緒的編號，進行各自的任務以及資料存取，過程示意如圖 3.2 所示。以 kernel_function<<<7, 8>>>(…)為例，在圖中總共開設 7 個 block 且每個 block 開設 8 個 thread 共 56 份任務緒，程式內容藉由不同執行緒的編號，各自運行與自己編號相同的任務。

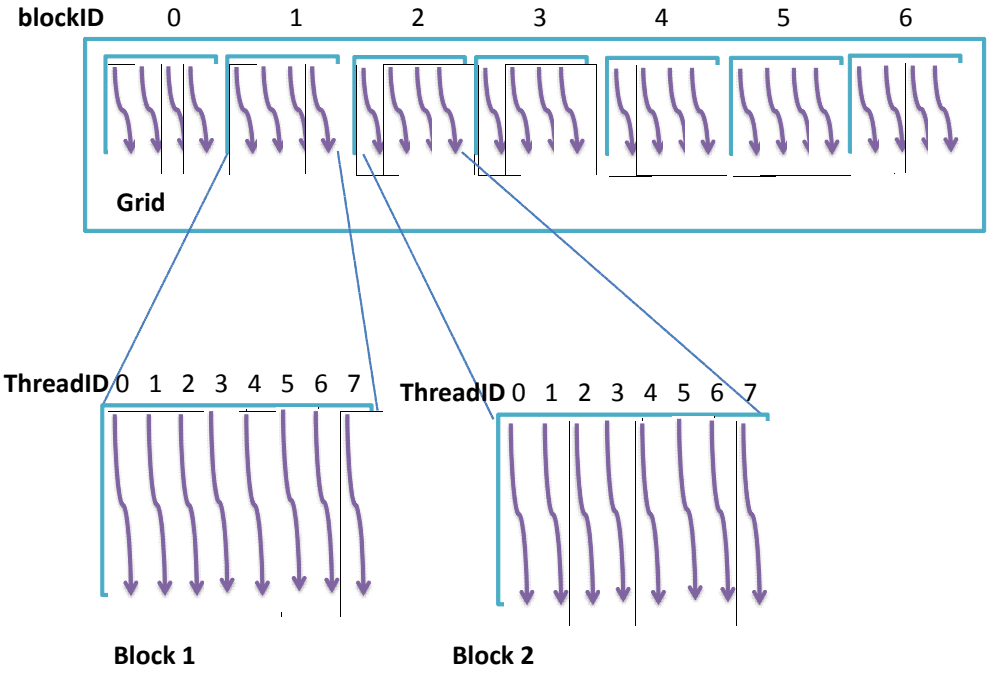


圖 3.2　block 大小為 7，thread 大小為 8 的執行緒區塊示意圖

GPU 卡上配置有多種不同的記憶體，程式平行化設計時須掌握記憶體特性配置資料，才能發揮平行的優勢。global memory 是 host 和 device 資訊溝通的記憶體，所有 block 皆可存取但存取，所耗時間是所有記憶體之冠。share memory 只在 block 內共用，不可跨 block 存取，host 程式也無法存取、溝通，share memory 讓 block 內部的 thread 互通資訊，存取時間較 global memory 快近百倍，但容量較小，程式撰寫時需搭配暫存器（register）的活用克服。register 是個別 thread 獨立存放資料的記憶體，在所有記憶體中存取速度最快，容量最小；通常用於存

放計算時執行緒的 ID。記憶體與 device 間的關聯關係如同圖 3.3 所示。不同 MP 依據不同 block 編號進行各自的任務，thread 間無法藉由 register 交換資訊，需要資料互通時得藉由 global memory 或 share memory 交換資訊。 block 間無法藉由 share memory 交換資訊，需要資料互通時得藉由 global memory 交換資訊。

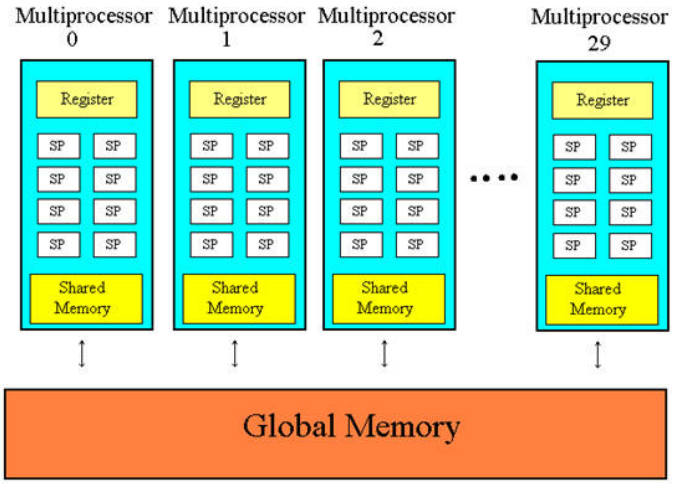


圖 3.3　記憶體與 device 間的關聯關係示意圖

3.4. SDAS 求解 TSP 的序列化演算程序設計

SDAS 求解旅行銷售員問題時，螞蟻逐步選擇城市繞行，完成一條符合條件的可行解。SDAS 在每代次中先將不同螞蟻依解建構的優劣分群，再依分群結果確定各費洛蒙項的優劣執行優加劣減的更新。TSP 的解是繞行序列，兩城市的連結 l_{ij} 代表城市 i, j 間路段的接續偏好，費洛蒙值 τ_{ij} 會視 l_{ij} 的優劣以添加或扣減。

SDAS 於演算開始時初始化所有城市連結 l_{ij} 上的費洛蒙值 τ_{ij}，以及迄今最佳解 S^* 及目標函數值 $f(S^*)$。城市 i 與城市 j 的轉移機率為

$$p_{ij}=\frac{\left(\tau_{ij}\right)^{\alpha}\cdot\left(\eta_{ij}\right)^{\beta}}{\sum_{l\in Z}\tau_{il}^{\alpha}\cdot\eta_{il}^{\beta}},\forall j\in\mathbf{Z}\ ,\qquad(3,12)$$

式中 **Z** 是目前的候選城市集合（尚未繞行的城市）； α 和 β 是螞蟻在選擇下一座城市 j 時對於費洛蒙強度及啟發資訊偏好的強度，當 α 越高時，螞蟻解建構時較看重過往的學習經驗，即費洛蒙的強度；相對地當 β 越高時，螞蟻會優先選擇較近的城市，啟發值 η_{ij} 設為城市 i 與 j 間距離的倒數，即 $1/c_{ij}$ 。啟發項的功能在引導優先選擇距離目前城市 i 最近的城市。所有螞蟻各自建構完解後，以式(3,1)計算目標函式值。SDAS 將螞蟻依 3.2 節的較佳較差界限法區分成優蟻、劣蟻、及普通螞蟻。並參照這些螞蟻解的城市連結進一步區分成優段和劣段，以執行 SDAS 擇段費洛蒙更新法。由第 3.2.2 所提出的「隨機差集比對」出來的優段集合 **A** 及劣段集合 **D** 後，添加優段費洛蒙

$$\tau_{ij}\leftarrow\tau_{ij}+\frac{1}{f\left(S^{*}\right)}\quad,\text{if } l_{ij}\in\mathbf{A}\qquad(3,13)$$

強化優段上的費洛蒙值。同時扣減劣段費洛蒙

$$\tau_{ij}\leftarrow\tau_{ij}\left(1-\rho\right)\quad,\text{if } l_{ij}\in\mathbf{D}\qquad(3,14)$$

弱化劣段上的引導。式中沿用使用者自訂的費洛蒙蒸發率 ρ 進行扣減，其值介於 0 與 1 之間。當優段和劣段費洛蒙項添加及扣減後，所有的費洛蒙矩陣值進行蒸發扣減，迄今最佳解的費洛蒙項 **G** ，進行額外的費洛蒙添加：

$$\tau_{ij}\leftarrow\tau_{ij}\left(1-\rho\right)+\Delta\tau_{ij}\ ,\forall l_{ij}$$
$$\text{where }\Delta\tau_{ij}=\begin{cases}\dfrac{1}{f\left(S^{*}\right)} & ,\text{if } l_{ij}\in\mathbf{G}\\ 0 & ,\text{otherwise}\end{cases}\ 。\qquad(3,15)$$

最後檢查當代次螞蟻最佳解S'是否優於迄今最佳解S^*。若是則更新$f(S^*)$和S^*為$f(S')$及S'，演算流程如下所示：

<u>序列化 SDAS 的演算流程</u>

1 設定所有城市連結l_{ij} 上的費洛蒙值為初始值。

2 設定求解迄今最佳解為空矩陣及螞蟻的目標函數值為無限大，即$S^* \leftarrow \langle\ \rangle$，$f(S^*) \leftarrow \infty$ 。

3 for $k \leftarrow 1$ to max_cycles

4 所有螞蟻依據轉移機率矩陣逐步選擇城市繞行。

5 執行特定的區域搜尋法(選擇性執行)。

6 計算各解的目標函式值。

7 將螞蟻依解的表現分類。

8 執行 SDAS 優加劣減擇段費洛蒙更新法 。

9 If 當代次螞蟻最佳解S'優於迄今最佳解

10 $f(S^*) \leftarrow f(S')$；$S^* \leftarrow S'$；

11 end if

12 end for

3.5. SDAS 用於 TSP 平行化設計－途程解建構

途程解建構(Tour construction)以及費洛蒙更新(pheromone)是蟻拓演算法的核心也是耗費演算資源最多的部分。當個人電腦的平行化程式執行環境(如：CUDA)成熟後，便成為蟻拓法平行化的研究重點，因蟻拓法是序列地"逐步"建構解，它的序列化演算本質，不易開發平行化程式。本研究參考 Cecilia(Cecilia, Garcia, Ujaldon, Nisbet, & Amos, 2011)等人提出的解建構平行化策略，在本節中

提出本研究擬定的平行化 SDAS 演算法求解 TSP 問題的兩種策略：atomic 及 partition。

3.5.1. 途程解建構 kernel－設計概念

典型的序列化解建構的架構中，於每一代次的解建構階段，每一隻螞蟻皆須計算n個城市間繞行任兩城市的選取機率，一般而言使用隨機輪盤法，派遣蟻次m隻下的解建構的演算時間複雜度為$O\left(\frac{mn(n+1)}{2}\right)$，更新費洛蒙矩陣的時間複雜度為$O(n^2)$，如此龐大的計算量在進行序列化演算時隨著問題的維度增加，演算所需耗費的時間呈等比遽增，在處理高維度 TSP 問題時，勢必無法兼顧求解品質與演算速度。

本研究全程以 Data-based 演算方式建構途程解，在更新費洛蒙矩陣方面，本研究提出 atomic 及 partition 兩種演算策略。SDAS 會依據當代次的優段集合**A**及劣段集合**D**更新費洛蒙矩陣，本研究提出城市連結標誌矩陣**T**來記錄城市連結屬於優段、劣段、或普通段集合，如圖 3.4 範例所示$n=4$的城市連結標誌矩陣**T**中$\{l_{13},l_{31},l_{43},l_{34}\}\in\mathbf{A}$、$\{l_{14},l_{41},l_{24},l_{42}\}\in\mathbf{D}$。

$$\mathbf{T}=\begin{bmatrix} 0 & 0 & \mathbf{A} & \mathbf{D} \\ 0 & 0 & 0 & \mathbf{D} \\ \mathbf{A} & 0 & 0 & \mathbf{A} \\ \mathbf{D} & \mathbf{D} & \mathbf{A} & 0 \end{bmatrix}_{4\times 4}$$

圖 3.4　城市連結標誌矩陣**T**範例

不同於傳統蟻拓法直接參照螞蟻的解更新費洛蒙矩陣，SDAS 將螞蟻分群後，將城市連結根據螞蟻分群的結果記錄在城市連結標誌矩陣 **T**， SDAS 間接地利用矩陣 **T** 對費洛蒙矩陣進行更新。

本節介紹本研究所提出平行化演算流程中，建構途程解的 kernel，此 kernel 的主要任務有二：首先是 Data-based 建構途程解，其次為配置城市連結標誌矩陣。

Data-based 建構途程解平行化流程

Data-based 的平行化主在將螞蟻選擇下一個候選城市建構路徑時執行的隨機式輪盤法選擇演算流程平行化。具體做法是應用 reduction 的平行演算程序大幅加速繞行城市時所需要的演算時間。執行 kernel 時在 Task-based 模式下的 thread 數量所對應的是螞蟻數，每個 thread 負責每隻螞蟻建構解的演算任務；相對地在 Data-based 模式下的 block 數量所對應的是螞蟻數，thread 數是城市的數量，每個 thread 負責解建構時每座城市個別的演算任務。原序列化演算在利用隨輪盤法建構路徑時，利用迴圈將每座城市的轉移機率一一跟隨機亂數值做比較，Task-based 模式下的演算模式也是承襲這樣的流程，相對於序列化只是能夠同時有數隻螞蟻同時建構繞行路徑，而相對的 Data-based 模式解建構時將原序列模式的隨機式輪盤法平行化，所有的城市都會產生一筆參考了轉移機率 p_{ij} 的隨機亂數值，並使用 reduction max 演算技巧兩兩一組同時平行比較後得出候選的城市，加速解建構的演算，不再需要一一再個別與亂數輪盤值比較。

最後在所有螞蟻繞行完成後，依 3.2 節的較佳較差界限法區分成優蟻、劣蟻、及普通螞蟻，同一 kernel 進入配置城市連結標誌矩陣演算流程。

配置城市連結標誌矩陣平行化流程－atomic 方法

將螞蟻根據繞行距離的表現用優、劣蟻做區別後，將城市連結優劣資訊如圖 3.4 所示存入城市連結標誌矩陣 **T** 中。執行 kernel 程式演算時，若同時有兩條以上的執行緒對同一個記憶體空間作寫入的動作，將導致競爭危害（race condition），造成運算結果出錯致使城市標誌矩陣無法正確進行演算。本研究應用普遍被應用於更新費洛蒙矩陣的 atomic function 來對城市標誌矩陣進行配置，但 atomic function 在運行的時候為確實迴避競爭危害，將演算回歸序列化，無法確實加速。

配置城市連結標誌矩陣平行化流程－partition 方法

在原資料結構下，必須引入 atomic function 以避免演算錯誤，為求確實平行化，本研究提出 partition 方法，改變城市連結標誌矩陣 **T** 的演算模式，使其能夠在不借助 atomic function 的下確實平行化。具體作法是螞蟻根據繞行距離的表現用優、劣蟻做區別後，將城市連結優劣資訊如圖 3.5 範例所示存入各自的城市連結標誌矩陣 $\mathbf{T}_m$ 中，之後將資料交付更新費洛蒙矩陣 kernel 進行演算。範例中 $\mathbf{T}_1$ 為優蟻的城市連結標誌矩陣、$\mathbf{T}_2$ 為劣蟻的城市連結標誌矩陣、$\mathbf{T}_3$ 為普通蟻的城市連結標誌矩陣，更新費洛蒙矩陣的 kernel 將同時參看所有的城市連結標誌矩陣決定更新費洛蒙值的策略。

$$\mathbf{T}_1=\begin{bmatrix} 0 & \mathbf{A} & \mathbf{A} & 0 \\ \mathbf{A} & 0 & 0 & \mathbf{A} \\ \mathbf{A} & 0 & 0 & \mathbf{A} \\ 0 & \mathbf{A} & \mathbf{A} & 0 \end{bmatrix}_{4\times4} \quad \mathbf{T}_2=\begin{bmatrix} 0 & 0 & \mathbf{D} & \mathbf{D} \\ 0 & 0 & \mathbf{D} & \mathbf{D} \\ \mathbf{D} & \mathbf{D} & 0 & 0 \\ \mathbf{D} & \mathbf{D} & 0 & 0 \end{bmatrix}_{4\times4} \quad \mathbf{T}_3=\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{4\times4}$$

圖 3.5　城市連結標誌矩陣 $\mathbf{T}_m$ 範例

3.5.2. 途程解建構 kernel－程式設計

本研究研擬的 SDAS 求解n－城市 TSP 時，在每代次演化初始，m隻螞蟻需先在 GPU 記憶體中配置$m\times n$個 8 位元整數的 global memory，用來儲存m隻螞蟻各自的陣列解$\mathbf{S}$，在計算接續城市的選擇機率也配置$n\times n$個 8 位元整數的 global memory。所有解建構完後 kernel 評估解品質分出優蟻、劣蟻。在 partition 方法中下一階段的費洛蒙更新平行化演算時每隻螞蟻皆配有各自的費洛蒙更新標誌，以矩陣設定，因此額外配置$m\times n\times n$個 8 位元整數的 global memory 儲存城市連結標誌矩陣$\mathbf{T}_{partition}$。另外在 atomic 方法中則是配置$n\times n$個 8 位元整數的 global memory 儲存城市連結標誌矩陣$\mathbf{T}_{atomic}$，此外為紀錄迄今的最佳解S^*及目標函數值$f(S^*)$，再配置$n+1$個 8 位元整數型態的 global memory 記憶體空間。最後需要配置$n\times n$個 8 位元整數型態的 global memory，儲存問題中城市間的距離矩陣ϕ。

Data-based 平行演算改善 Task-based 模式的解建構輪盤法演算。Task-based 模式中每個 thread 均需執行反覆迴圈式序列化的演算以決定下一座城市。相對地本研究的 Data-based 模式在每一個 thread 上各自產生亂數值(本研究使用 cuda 函式庫的 curand 在 GPU 環境中執行)即可進行平行式的 reduction 運算，配置 block 數量等於螞蟻數量m，thread 配置數量等於城市數量n。

Data-based 平行演算善用 share memory 的高速存取效能，將演算需要的資料的值儲存在 share memory 支援更快捷的運算，在輪盤法建立候選清單時，配置$4\times n$個 8 位元整數的 share memory 輔助演算。

至此運作解建構平行化的 kernel

ConstructSolutionPath <<< $m, n, 4n$ >>>
(*route* **S**,*transition* **P**,*Tpheromone* **T**,*int** S*, *float** f(S*))

在執行時所需的 global memory 和 share memory 以及 block 數和 thread 數皆配置完成。

Data-based 模式解建構平行演算程序

原始的輪盤法中，從城市 i 建構候選城市 j 時會藉由式(3,12)得出轉移機率值 p_{ij}，從城市 i 起始建構城市 j 時，給一介於 0 與 1 之間的亂數值，逐一與各個城市 j 的 p_{ij} 比較，進而判斷落入哪一個城市 j 選中的數值居間。

在 Data-based 模式下，將逐一比較的方式平行化，每座城市的 p'_{ij} 不再是與輪盤值反覆地做序列化比較，相對地將亂數輪盤值與原序列化解建構的轉移機率 p_{ij} 直接參與輪盤值的計算，因此每座候選城市計算出來的 p'_{ij} 便可以使用 reduction max 快速選出城市 j。其中

$$p'_{ij} = p_{ij} \cdot q_j \cdot r_j \tag{3,16}$$

q_j 為 tabu value，若在解建構過程中已經過的城市其值為 0，否則為 1，如此 p'_{ij} 對已建構的城市 j 而言為 0，確保可行解的建構，r_j 為介於 0 到 1 之亂數值，同序列化輪盤法輪盤值的概念，確保其隨機性以避免過早陷入區域最佳解。由(3,16)中得到的 p'_{ij} 使用 reduction max 求出螞蟻所選中建構的下一座城市

$$j = \max\left\{p'_{ij}, j \in [1, \mathrm{n}]\right\}。 \tag{3,17}$$

Data-based 平行化演算中不同城市 i 與 j 的轉移機率 p'_{ij} 使用不同的 thread 計算，每一個 thread 單獨計算一個 p'_{ij}，並於式(3,17)作 reduction max 將所建構之城市 j 選出，在 thread 的運用上彰顯出來平行化的優勢，顯著的增進平行化資源運用及演算效能。

平行化 reduction max 的演算流程如圖 3.6 所示，設一矩陣大小為 8，step1 啟用 $\lceil 8/2 \rceil$=4 個 thread 進行數值比較，每個 thread 皆只做一次比較，第一次比較時需囊括所有的值，最後每個 thread 比較完後皆留下比較的結果，需要參看的值由原本的 8 個降為 4 個，邁向下一個 step 時便只需要原來需要 $\lceil 4/2 \rceil$=2 個 thread 來比較，依此類推，比較 8 次才能找出最大值的計算便縮減到了 3 次。

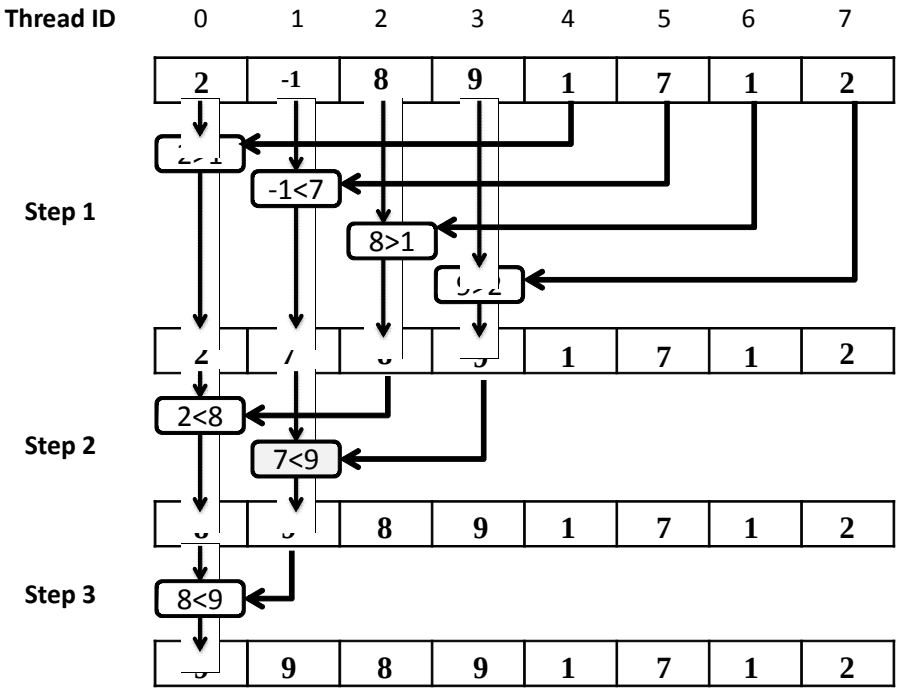


圖 3.6　Reduction max 示意

計算 p'_{ij} 時可利用配置在 kernel 程式的 share memory，一個 block 所分派的任務便是建構一隻螞蟻的解，透過 share memory 不同 block 資訊不共有的特點，可以對程式在建構解平行化的設計上能夠更為直覺，方便管理。

如圖 3.7 所示 p_{ij} 、r_j 以及 q_j 在不同城市 j 下計算 p'_{ij} 時皆由不同的 thread 所對應負責運算，以城市 1 來說，p_{i1}=0.1、r_1 =0.2、q_1=0，則由 ID 為 0 的 thread 計算可得 p'_{1j} =0。所有 thread 計算結束後在做 reduction max 選出候選城市，如圖例中選到的即為城市 n。在計算時 p_{ij}、r_j 、q_j、p'_{ij} 皆可利用配置於 share memory 的空間加速演算，詳細程式碼請參閱附錄：建構途程解 kernel 程式碼。

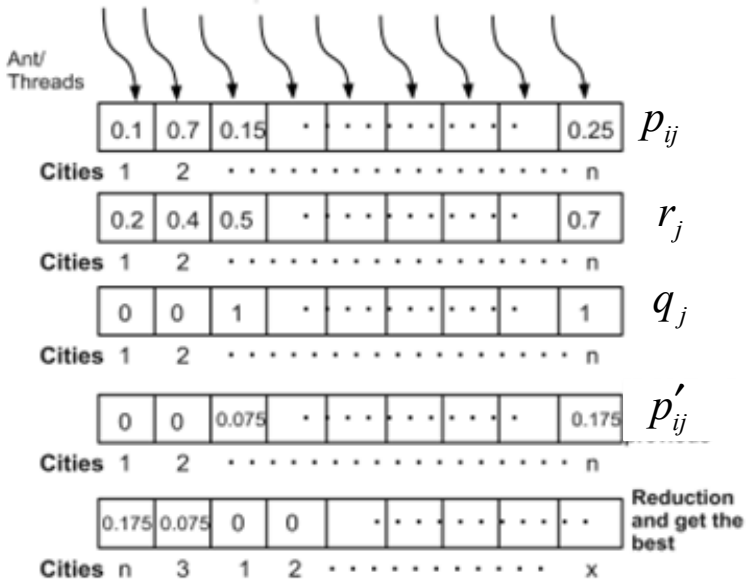


圖 3.7　Data-based 平行化演算示意

配置城市連結標誌矩陣平行演算程序－atomic 方法

城市連結標誌矩陣 $\mathbf{T}_{atomic}$ 根據螞根據 3.2 節計算出的優段連結集合 $\mathbf{A}$、劣段連結集合 $\mathbf{D}$ 分群的結果標誌，更新費洛蒙矩陣 kernel 便可依據 $\mathbf{T}_{partition}$ 上各城市連結所屬的分群做費洛蒙增加或扣減。在計算過程當中 $\mathbf{A}' = \mathbf{A}'' - \mathbf{G} - \mathbf{D}''$ 和 $\mathbf{D}' = \mathbf{D}'' - \mathbf{G} - \mathbf{A}''$ 的計算由於 $\mathbf{A}''$ 與 $\mathbf{D}''$ 並非互斥，因此有可能在做集合元素分群的運算時會有多對一寫入的衝突，在序列化的程式中，可利用 if-else 函式一一判別再寫入，但平行化程式因存在競爭危害，必須藉由 atomic function 或重新設計資料結構才得以運行。

在標誌時不同 block 處理不同螞蟻的任務，不同 thread 負責標誌不同繞行順位下的城市連結。如 block ID 為 0、thread ID 為 3 的執行緒即是負責處理螞蟻編號為 0 所建構的解中繞行順位為 3 與 4 間的城市連結 l_{34} 與 l_{43}，如 block ID 為 3、thread ID 為 n 的執行緒即是負責處理螞蟻編號為 0 所建構的解中繞行順位為 n 與 0 間的城市連結 l_{n0} 與 l_{0n}。

atomic function 的運作機制為，當不同 thread 都在處理同為優蟻或劣蟻的解，並在同一時間對 $\mathbf{T}_{atomic}$ 的同一個元素做標誌，此時 atomic function 機制便會啟用，強制將此處的更新演算流程序列化以防止演算出錯。

標誌矩陣初始值 $\mathbf{T}_{atomic} = \begin{pmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{pmatrix}$。較佳螞蟻中，其優蟻上的城市連結 $l_{ij}^{\mathbf{B}}$ 所對應到的 $\mathbf{T}_{atomic,ij}$ 在標誌矩陣中便標記為 2，而較差螞蟻中，其劣蟻上的城市連結 $l_{ij}^{\mathbf{W}}$ 所對應到的城市連結 $\mathbf{T}_{atomic,ij}$ 在標誌矩陣中便標記為 0。本法每一個 thread 在進行標記時的計算邏輯為

1. $1 \oplus 1 = 1$：若標誌皆沒有入圍較佳或較差螞蟻，則標記 $\mathbf{T}_{atomic,ij} = 1$。

2. $1 \oplus 2 = 2, 2 \oplus 1 = 2$：若任一標誌入圍較佳螞蟻，則標記 $\mathbf{T}_{atomic,ij} = 2$。

3. $1 \oplus 0 = 0, 0 \oplus 1 = 0$：若任一標誌入圍較差螞蟻，則標記 $\mathbf{T}_{atomic,ij} = 0$。

4. $0 \oplus 2 = 3, 2 \oplus 0 = 3$：若一標誌入圍較差螞蟻，一標誌入圍較佳螞蟻，則標記 $\mathbf{T}_{atomic,ij} = 3$。

5. $3 \oplus \text{any} = 3, \text{any} \oplus 3 = 3$：若已標誌互斥，$\mathbf{T}_{atomic,ij} = 3$。

配置城市連結標誌矩陣平行演算程序－partition 方法

因為在 $\mathbf{T}_{atomic}$ 中的城市連結標誌存在被同一個 thread 存取的可能，因此需要利用 atomic function 防止競爭危害的發生，因此本研究提出 partition 方法，配置相同於派遣蟻數 m 的城市連結標誌矩陣 $\mathbf{T}_{partition}$ m 個，每隻螞蟻有獨立的矩陣可供標誌，由於螞蟻在解建構時城市不會重複繞行，因此能夠確保競爭危害不會發生。

在標誌時不同 block 處理不同螞蟻的任務，同時也會隨著 block ID 不同，標誌的 $\mathbf{T}_{partition}$ 也不一樣，不同 thread 負責標誌不同繞行順位下的城市連結。

各標誌矩陣初始值 $\mathbf{T}^{k}_{partition} = \begin{pmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{pmatrix}, k \in \{1, 2, ..., m\}$。本法定義優蟻上的城市連結 $l^{\mathbf{B}}_{ij}$ 所對應到的 $\mathbf{T}^{B}_{partition,ij}$ 在標誌矩陣中便為較佳城市連結標誌，標誌

$\mathbf{T}^{\mathbf{B}}_{partition,ij}=3, l_{ij} \in \mathbf{A''}$，本法又定義劣蟻上的城市連結 $l^{\mathbf{W}}_{ij}$ 所對應到的 $\mathbf{T}^{\mathbf{W}}_{partition,ij}$ 在標誌矩陣中便為較佳城市連結標誌，標誌 $\mathbf{T}^{\mathbf{W}}_{partition,ij}=-1, l_{\mathrm{ij}} \in \mathbf{D''}$。

3.6. SDAS 用於 TSP 平行化設計－更新費洛蒙矩陣

完成螞蟻解建構並且更新迄今最佳解，且城市連結標誌矩陣標記完成後，須執行蟻拓法的費洛蒙值更新。相較於解建構，更新費洛蒙的流程相對單純。在本章將針對本研究所提出的 atomic 及 partition 方法做介紹。

典型的序列化蟻拓法下，所有螞蟻共用同一個費洛蒙矩陣。本研究針對 SDAS 的費洛蒙更新機制研擬適合平行化的資料結構型態，在本節提供提供費洛蒙矩陣更新的平行化方式，，並於第四章做進一步的數據分析。

3.6.1. 更新費洛蒙矩陣 kernel

本研究研擬的 SDAS 更新費洛蒙矩陣法下，將參照 3.5 節中的城市連結標誌矩陣進行平行化演算，個別單一的費洛蒙值分別由不同的執行緒處理。

費洛蒙更新的 GPU 記憶體配置及 kernel 平行程式設計

SDAS 求解 TSP 更新費洛蒙矩陣時，在 partition 每隻螞蟻皆配有各自的費洛蒙更新標誌，以矩陣設定，因此配置 $m \times n \times n$ 個 8 位元整數的 global memory 儲存城市連結標誌矩陣 $\mathbf{T}_{partition}$。若在 atomic 方法中則是配置 $n \times n$ 個 8 位元整數的 global memory 儲存城市連結標誌矩陣 $\mathbf{T}_{atomic}$。費洛蒙矩陣 $\mathbf{R}$ 也配置 $n \times n$ 個 8 位元整數的 global memory。此外紀錄迄今的最佳解 S^* 及解的目標函數值 $f\left(\mathrm{S}^*\right)$ 也同時在額外添加迄今最佳解的費洛蒙時需要配置 $n+1$ 個 8 位元整數的 global memory。

利用標誌矩陣更新費洛蒙矩陣平行化的演算過程中，每一個 thread 都對應到標誌矩陣及費洛蒙矩陣中個別獨立的元素。在 partition 與 atomic 方法中階配置 block 數量為 $\left\lceil \frac{n^2}{1024} \right\rceil$、thread 數量為 1024，其中 1024 為本研究的硬體設備門檻(最大每個 block 可配置的 thread 數量)。不難觀察出 atomic 方法在費洛蒙值更新上所需耗費的時間資源，較低於 partition 方法，但儘管如此這也是在解建構 kernel 中使用 atomic function 所換來的優勢，因此孰優孰劣單就個別 kernel 的設計，無法直觀看出。

在標誌矩陣更新費洛蒙矩陣平行化的演算過程，由於 share memory 可使用的空間無法支應演算時所需要的龐大資料量，本 kernel 不借助 share memory 的演算效能。

至此運作費洛蒙矩陣更新的 kernel

$$PheromoneRefresh <<< \left\lceil \frac{n^2}{1024} \right\rceil, 1024, 0 >>>$$
$$(pheromone\ \mathbf{R}, Tpheromone\ \mathbf{T}, int^*\ \mathrm{S}^*, float^*\ f(\mathrm{S}^*))$$

在執行 kernel 程式時所需的 global memory 以及 block 數和 thread 數皆已配置完成。

atomic 方法更新費洛蒙矩陣

本法參看 $\mathbf{T}_{atomic}$ 上的標誌對相對應位置的費洛蒙值 τ 進行更新，不同的費洛蒙直皆平行由不同的 thread 負責，如 τ_{ij} 需參看 $\mathbf{T}_{atomic,ij}$ 的標誌，任兩組不同的 i, j 皆由不同的執行緒演算，每一個 thread 在進行任務時的計算邏輯為

1. 若 $\mathbf{T}_{atomic,ij}=1$ 或 $\mathbf{T}_{atomic,ij}=3$：該城市連結不屬於優段集合 **A** 亦不屬於劣段集合 **D**，蒸發一次該費洛蒙值，$\tau_{ij} \leftarrow \tau_{ij}(1-\rho)$ 。

2. 若 $\mathbf{T}_{atomic,ij}=2$：該城市連結屬於優段集合 **A**，蒸發一次該費洛蒙值，增加一次該費洛蒙值，$\tau_{ij} \leftarrow \left(\tau_{ij}+\frac{1}{f(S^*)}\right)(1-\rho)$ 。

3. 若 $\mathbf{T}_{atomic,ij}=0$：若任一標誌入圍較差螞蟻，則 $\chi=0$，蒸發一次該費洛蒙值後額外再蒸發一次費洛蒙值，$\tau_{ij} \leftarrow \tau_{ij}(1-\rho)\left(1-\rho^{special}\right)$ 。

費洛蒙矩陣中所有元素皆做完計算後，還原各螞蟻標誌矩陣初始值 $\mathbf{T}_{atomic}=\begin{pmatrix}1 & \cdots & 1\\ \vdots & \ddots & \vdots\\ 1 & \cdots & 1\end{pmatrix}$。並於最後強化迄今最佳解城市連結的費洛蒙值

$$\tau_{ij} \leftarrow \tau_{ij}+\Delta\tau_{ij}$$

$$where\ \Delta\tau_{ij}=\begin{cases}\frac{1}{f(S^*)} & ,\text{if } l_{ij}\in \mathbf{G}\\ 0 & ,\text{otherwise}\end{cases}$$ 。

partition 方法更新費洛蒙矩陣

本法參看 $\mathbf{T}^k_{partition}$ 上的標誌對相對應位置的費洛蒙值 τ 進行更新，不同的費洛蒙直皆平行由不同的 thread 負責，如 τ_{ij} 需參看 $\mathbf{T}^k_{partition}, k\in\{1,2,...,m\}$ 的標誌，任兩組不同的 i, j 皆由不同的執行緒演算，本研究定義每一個 thread 在進行任務時的計算決策值 $\mathbf{T}_{partition,ij}=\mathbf{T}^1_{ij}+\mathbf{T}^2_{ij}+...+\mathbf{T}^m_{ij}$。上一節的例子中 $\mathbf{T}^{\mathbf{B}}_{ij}=3$、$\mathbf{T}^{\mathbf{W}}_{ij}=-1$， 每一個 thread 的計算結果如下

1. 於此例下若城市連結 l_{ij} 同時存在於較佳螞蟻與較差螞蟻中，且存在的連結數量相等或城市連結沒有入圍決策時， $\mathbf{T}_{partition,ij}=m$ 。此城市連結視為普通城市連結，蒸發一次該費洛蒙值，$\tau_{ij}\leftarrow\tau_{ij}(1-\rho)$ 。

2. 於此例下若城市連結 l_{ij} 存在於較佳螞蟻的連結數大於存在於較差螞蟻的連結數，$\mathbf{T}_{partition,ij}>m$ ，此城市連結視為較佳城市連結。蒸發一次該費洛蒙值，增加一次該費洛蒙值，$\tau_{ij}\leftarrow\left(\tau_{ij}+\frac{1}{f(S^*)}\right)(1-\rho)$ 。

3. 於此例下若城市連結 l_{ij} 存在於較差螞蟻的連結數大於存在於較佳螞蟻的連結數， $\mathbf{T}_{partition,ij}<m$ ，此城市連結視為較差城市連結。蒸發一次該費洛蒙值後額外再蒸發一次費洛蒙值，$\tau_{ij}\leftarrow\tau_{ij}(1-\rho)(1-\rho^{special})$ 。

費洛蒙矩陣中所有元素皆做完計算後，還原各螞蟻標誌矩陣初始值 $\mathbf{T}^k=\begin{pmatrix}1 & \cdots & 1\\ \vdots & \ddots & \vdots\\ 1 & \cdots & 1\end{pmatrix}, k\in\{1,2,...,m\}$ 。並於最後強化迄今最佳解城市連結的費洛蒙值

$$\tau_{ij}\leftarrow\tau_{ij}+\Delta\tau_{ij}$$

$$where\ \Delta\tau_{ij}=\begin{cases}\frac{1}{f(S^*)} & ,\text{if } l_{ij}\in\mathbf{G}\\ 0 & ,\text{otherwise}\end{cases}$$ 。

在 partition 方法中，$\mathbf{T}_{ij}^{\mathbf{B}}$ 、$\mathbf{T}_{ij}^{\mathbf{W}}$ 與 χ_{ij} 可由使用者自行設定以滿足各樣不同決策模式，如: $\mathbf{T}_{ij}^{\mathbf{B}}=5$ ，$\mathbf{T}_{ij}^{\mathbf{W}}=-1$ 則較佳螞蟻的城市連結對於決策的影響則大於較

差螞蟻的城市連結，相對地若 $\mathbf{T}_{ij}^{B}=1$，$\mathbf{T}_{ij}^{W}=-5$ 則較差螞蟻的城市連結對於決策的影響則大於較佳螞蟻的城市連結。

3.7. 小結

本章為平行化 SDAS，針對 SDAS 演算的結構，採用 data-based 途程解建構方法，並在途程解建構時進行城市連結標誌矩陣的標誌紀錄，根據不同的城市連結標誌矩陣的標誌方式分成了 atomic 方法及 partition 方法。本研究針將部分的費洛蒙矩陣更新演算流程從費洛蒙矩陣更新的 kernel 移到了途程解建構的 kernel。相較一般蟻拓法把所有的費洛蒙矩陣皆在同一個 kernel 內完成的平行化方式，本研究認為在 CUDA 的程式架構下，本研究所提供的平行化方式在演算上更有優勢。

驗證數據與不同策略下的演算結果分析將於第四章做展示。

第四章　平行化 SDAS 及範例驗證

本章展示旅行銷售員問題的平行化 SDAS 求解系統。為驗證所提平行化的演算程序，本研究使用 TSPLIB 中的標竿問題，測試本研究中提出的 SDAS 各個演算階段下以及整體上的加速效能，再與相關研究做進一步的比較分析。

本研究開發的平行化 SDAS 求解系統，根據城市連結標記矩陣資料存取方式的不同，提出兩種方法：atomic 以及 partition。差別在更新標誌矩陣時是否啟用 CUDA 語言中的 atomic 存取機制。前者 atomic 模式啟用 atomic 並能維持城市連結集合互斥，後者 partition 沒有啟用 atomic 且並未維持城市連結互斥。

4.1. 平行化 SDAS 求解系統實作

本研究使用 Microsoft Visual Studio 2010 開發工具，在 Microsoft .NET Framework 平台下，以 Nvidia 的 CUDA 程式語言開發平行化 SDAS 演算的 TSP 求解系統。

表 4.1 是本研究演算時的硬體環境。使用的 GPU 是 Nvidia 的 Geforce 顯示卡系列，型號為 GTX680。內有 8 個 SM，每個 SM 中有 167 個計算核心（Core），總計算核心數量為167×8＝1536 個。在 CUDA 內每個 block 最多可以配置 1024 個 thread，而 block 的配置上限是 65535 個。將 thread 負責的運算送交 Core 做計算時每次可同時運送 32 個 thread，每個 block 有 48Kib 的 share memory，GPU 內全域共用記憶體（global memory）大小是 4096Mib。

表4.1.　範例測試執行求解系統的硬體環境

GPU	nVIDIA Geforce GTX680
每個 SM 中 Core 數量	167
SM 數量	8
總 SP 數量	1536
每個 block 中的 thread 最大值	1024
Block 最大值	65535
Warp 大小	32
每個 Block Share memory 大小	48Kib
Global memory 大小	4096Mib

4.2. 演算法成效分析

實例測試首在驗證及分析平行化 SDAS 的加速效能。雖然平行和序列演算的邏輯相同，都由電腦處理器運算，但平行化程式有不同的資料結構。首先比較沒有平行化的程式與平行化版間的差異，之後再將途程解建構和費洛蒙矩陣更新下所使用的的不同策略作做個別分析。然後進行演算法加速成效分析，最後再對不同的求解策略做進一步分析。

執行 SDAS 求解時的參數設定內容：$\alpha = 1$ 、$\beta = 3$ 、$\tau_0 = 0.5$ 、以及費洛蒙蒸發率 $\rho = 0.1$ 。優加劣減的管制界限幅度 $\varpi = 0.85$、優段門檻 $\theta^A = 0.8$、劣段門檻 $\theta^B = 0.6$、partition 演算策略的 $\mathbf{T}_{ij}^{\mathbf{B}} = 3$、$\mathbf{T}_{ij}^{\mathbf{W}} = -1$。

4.2.1. 平行化 SDAS 與 SDAS 之同質性比較

序列化的演算流程平行化後使用資料結構有不同。在展示加速成效前，本節先比較平行化及序列化的求解品質。

表 4.2 中比較序列 SDAS 與平行化 SDAS 求解不同 TSP 的求解效能。兩者皆不啟用區域搜尋，僅以標準的 SDAS 演算法求解 2500 代次，使用的螞蟻數等於城市數量。每個 TSP 反覆求解 15 次列出求得的最佳解的目標函數值平均及標準差。此外並做雙母體平均數 T 檢定 $H_0:\mu_0 \geq \mu_1$ 、$H_0:\mu_0 < \mu_1$ ，其中 μ_0 是平行化 SDAS 求得的類最佳解目標函數值平均，μ_1 是序列 SDAS 求得的類最佳解目標函數值平均。表第一欄是測試例題名稱；第二欄是已知最短距離；第三、四欄是序列 SDAS 求得的最短距離平均及標準差；第五、六欄是平行化 SDAS 使用 atomic 方法求得的最短距離平均及標準差；第七欄是 T 檢定的 p-value；第八到十欄是平行化 SDAS 使用 partition 方法求得的最短距離平均、標準差、及 T 檢定的 p-value。

表 4.3 是序列 SDAS、平行化 SDAS 使用 atomic 方法、及平行化 SDAS 使用 partition 方法三個版本演算的平均執行時間。

表4.2.　序列化 SDAS 與平行化 SDAS 於不同例題下的比較

測試例題	最短距離	序列化 SDAS		平行化 SDAS (atomic)		
		平均距離	標準差	平均距離	標準差	p-value
att48	10628	10851.9	193.46	10760.2	95.60	<0.05
kroC100	20750.76	22144.6	456.66	21084.4	158.46	<0.05
pcb442	50783.54	56773.8	736.84	54754.8	640.80	<0.05
pr1002	259045	326468	12890.28	312708.9	10672.37	<0.05

測試例題	最短距離	平行化 SDAS (partition)		
		平均距離	標準差	p-value
att48	10628	10692	51.78	<0.05
kroC100	20750.76	20921.70	134.1	<0.05
pcb442	50783.54	53170.44	442.49	<0.05
pr1002	259045	295467	7890.28	<0.05

表4.3.　序列化 SDAS 與平行化 SDAS 於不同例題演算的平均執行時間比較

測試例題	平均演算耗時 (ms)		
	序列化 SDAS	平行化 SDAS (atomic)	平行化 SDAS (partition)
att48	549.4	4461	3644
kroC100	3827.7	13051	14491
pcb442	227206.6	229548	277737
pr1002	12459500	2253500	2422379

本實驗觀察到在不啟動區域搜尋下，兩個平行演算模式的解品質與較序列 SDAS 優，且加速明顯，本研究認為，關鍵點在於解建構時隨機式輪盤法上的資料結構轉變所導致。兩個平行演算模式比較下 partition 方法雖然有更好的求解品質，但加速效能不及 atomic 方法。其中平行化 SDAS atomic 模式求解例題 kroC100 時演算時間低於複雜度較低的 att48，本研究認為是 CUDA 的 atomic 函式在小維度的問題中沒有辦法發揮優勢所致。

4.2.2. 各演算階段的加速結果比較

在經過途程解建構的 kernel 跟費洛蒙矩陣更新 kernel 的加速後，本節分別計算建構途程解與費洛蒙矩陣更新兩 kernel 個別的運算時間，以及分別於相對應的 kernel 在同樣的資料結構及運算邏輯下，計算序列化的運算時間，求得該 kernel 程式的 speed up，並與其他研究的數據作比較分析。定義 speed up factor

$$S = \frac{E\left[T_{serial}\right]}{E\left[T_{parallel}\right]} \quad , \qquad (4,1)$$

其中 T_{serial} 為演算在序列化程式的執行下所需耗費的時間，$E\left[T_{serial}\right]$ 為其期望值。相對地 $T_{parallel}$ 為演算在平行化程式的執行下所需耗費的時間，$E\left[T_{parallel}\right]$ 為其期望值。

建構途程解的 kernel 平行程式

建構途程解的平行演算程式是蟻拓法的主要運算。其演算速度影響求解效能高低。本研究進一步測試本研究所提出的 SDAS 平行運算的 atomic 與 partition 方法，以及 AS (Cecilia et al., 2011)平行演算下的 data-based 及 task-based 兩種模式的運算效能，表 4.4 列出這四種模式的加速結果。本測試設定螞蟻為城市數，求解 100 代次為停止條件，演算時間平均是這 100 代次中每代次所耗費的運算時間平均，比較耗用時間平均以及換算的 speed up factor。表 4.4 中第一欄是測試例題名稱；第二欄是測試的模式；第三欄是每一代次平行演算時間平均；第四欄是 SDAS 的序列演算耗用時間；第五欄是 speed-up。

表中 N/A 的內容，若非本研究所提出，係指文獻內沒有相關資料，若是本研究提出，則是因為在大維度問題下的記憶體不足程式無法執行完畢

表4.4.　建構途程解 kernel 加速比較

例題名稱	方法	平行演算耗時 (ms)	序列演算耗時 (ms)	Speed up
att48	SDAS atomic	1.824	2.348	1.68×
	SDAS partition	1.4007	2.407	1.31×
	AS data-based	0.34	1.02	3×
	AS task-based	13.14	N/A	N/A
krocC100	SDAS atomic	3.498	21.03	6.01×
	SDAS partition	4.293	22.07	5.14×
	AS data-based	0.91	11.83	13×
	AS task-based	56.89	N/A	N/A
pcb442	SDAS atomic	67.05	1800	17.65×
	SDAS partition	77.68	1891	16.28×
	AS data-based	36.57	731	20×
	AS task-based	N/A	N/A	N/A
d657	SDAS atomic	200.8	5881	29.29×
	SDAS partition	195.18	6151	31.51×
	AS data-based	123.17	2524	20.5×
	AS task-based	2770.32	N/A	N/A
rat783	SDAS atomic	501.3	10092	20.13×
	SDAS partition	368.25	10564	28.69×
	AS data-based	N/A	N/A	N/A
	AS task-based	N/A	N/A	N/A
pr1002	SDAS atomic	789.6	21013	26.31×
	SDAS partition	700.08	22130	31.61×
	AS data-based	417.72	12113	29×
	AS task-based	6181	N/A	N/A
pr2392	SDAS atomic	4886	285689	58.5×
	SDAS partition	N/A	N/A	N/A
	AS data-based	5461.06	152909	28×
	AS task-based	65537.7	N/A	N/ A

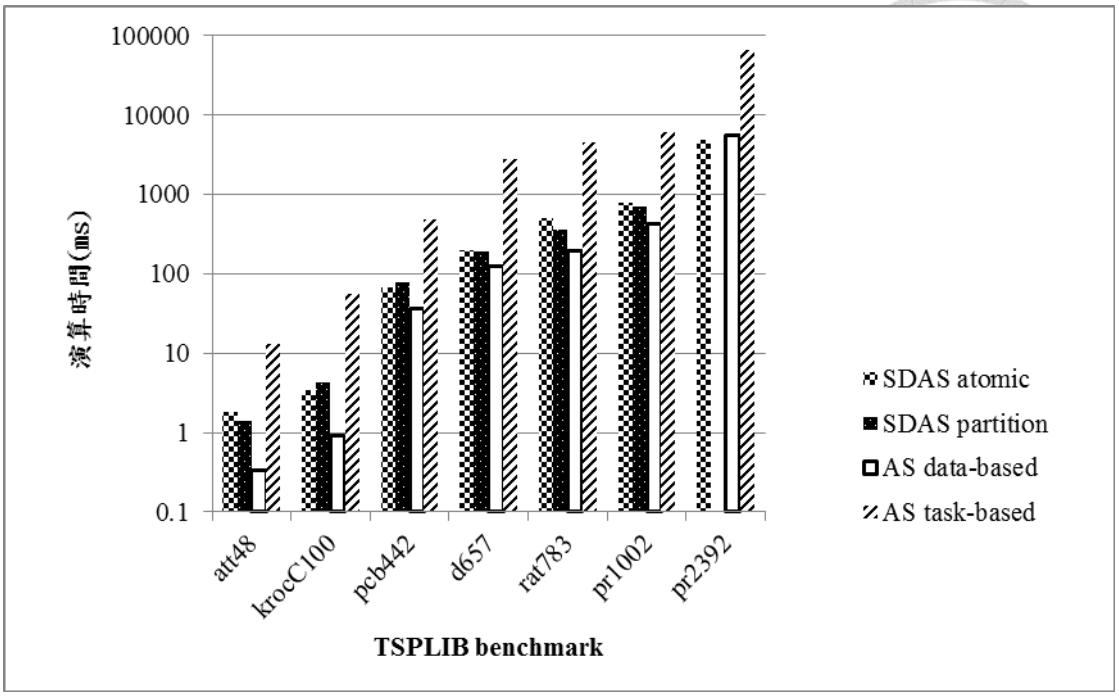


圖 4.1　建構途程解 kernel 的平行程式運算時間比較

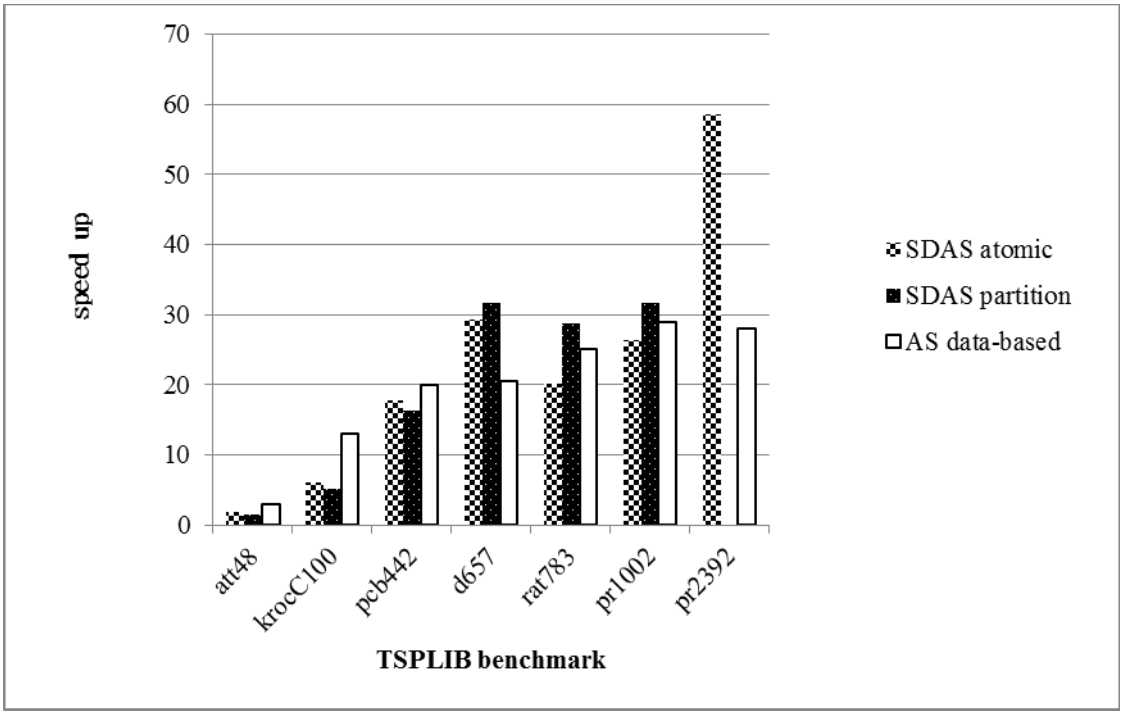


圖 4.2　建構途程解 kernel 的平行程式 speed up 比較

在低維度的 TSP 下 AS data-based 方法在演算時間與 speed up 皆優於本研究所提出的兩種方法，但在高維度下 SDAS atomic 在演算時間上與 speed up 皆有較佳的表現。本研究認為這樣的現象來自於 atomic 方法中的配置城市連結標誌矩陣的加速效能在大維度下方始展現，相較 AS data-based 方法與本研究所提出的 SDAS partition 下，SDAS atomic 方法途程解建構的 kenel 在處理大維度的運算時間時較為穩健。

另外在序列化演算耗費時間可以看出本研究的序列化演算所耗費時間普遍較 AS data-based 方法久，但隨著維度的提增，speed up 與演算時間皆可與之披敵，顯示本研究為平行化所做資料結構上的改變較 AS data-based 適合用於平行化演算。

更新費洛蒙矩陣的 kernel 平行程式

類似地，本研究針對費洛蒙更新演算部分的平行程式進行效能比較，求算四種模式的耗用時間，表 4.5 列出這四種模式在費洛蒙更新演算時的耗用時間，及速效果。本測試設定螞蟻數為城市數量，求解 100 代次為停止條件，演算時間平均為 100 代次中每代次所耗費的時間平均。統計各模式演算時間平均並計算相對於序列模式的 speed-up。表 4.5 中第一欄是測試例題名稱；第二欄標示各模式的執行方法；第三欄是每一代次平行演算平均所耗用時間；第四欄是序列演算耗用時間；第五欄為平行模式相較序列模式的 speed-up。

表4.5.　更新費洛蒙矩陣 kernel 演算耗費時間及加速比

例題名稱	方法	平行演算耗時 (ms)	序列演算耗時 (ms)	Speed up
att48	SDAS atomic	0.029	0.054	1.85×
	SDAS partition	0.044	1.145	3.3×
	AS atomic	0.04	0.1	2.5×
	AS StoG	0.66	N/A	N/A
krocC100	SDAS atomic	0.033	0.078	2.33×
	SDAS partition	0.136	0.653	4.79×
	AS atomic	0.09	0.36	4×
	AS StoG	4.5	N/A	N/A
pcb442	SDAS atomic	0.067	0.25	5.59×
	SDAS partition	15.1	63.8	6.59×
	AS atomic	0.79	N/A	7.5×
	AS StoG	1555.03	N/A	N/A
d657	SDAS atomic	0.104	0.58	5.58×
	SDAS partition	54.3	358	6.81×
	AS atomic	1.85	5.93	11×
	AS StoG	7547.1	N/A	N/A
rat783	SDAS atomic	0.152	0.92	6.48×
	SDAS partition	95.6	619.57	6.48×
	AS atomic	N/A	N/A	N/A
	AS StoG	N/A	N/A	N/A
pr1002	SDAS atomic	0.199	3.16	15.89×
	SDAS partition	185.3	1263.33	6.82×
	AS atomic	4.22	N/A	12.5×
	AS StoG	40977.3	N/A	N/A
pr2392	SDAS atomic	0.97	24.3	25.05×
	SDAS partition	N/A	N/A	N/A
	AS atomic	N/A	N/A	19×
	AS StoG	N/A	N/A	N/A

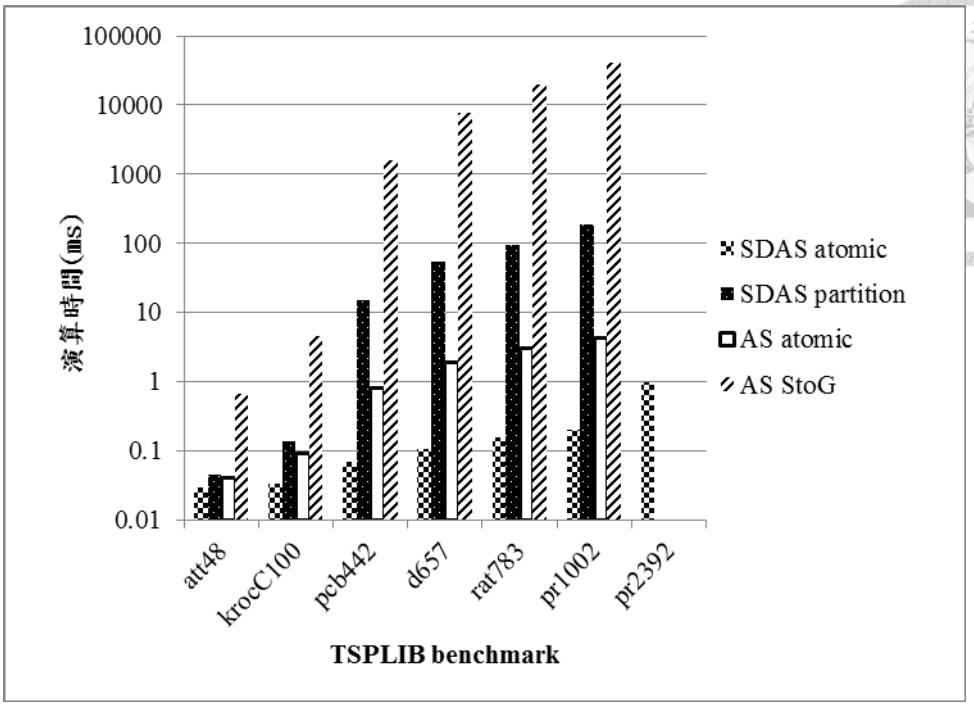


圖 4.3　更新費洛蒙矩陣 kernel 的演算時間比較

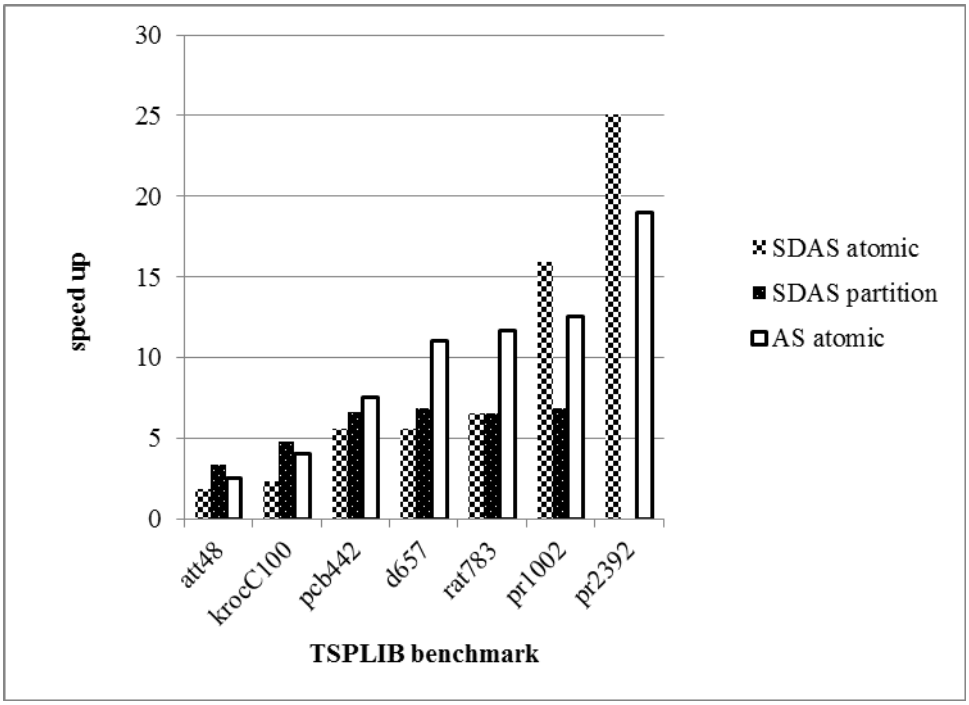


圖 4.4　更新費洛蒙矩陣 kernel 的 speed up 比較

本研究將配置城市連結標誌矩陣的工作留至途程解建構做演算，因此在本研所提出的更新費洛蒙矩陣的 kernel 只需要參照在標誌矩陣中對應的數值，便能夠快速完成更新費洛蒙值的演算，特別是 SDAS atomic 方法下演算的平均耗費時間獨佔鰲頭。但在 SDAS partition 方法中因為演算資訊隨問題維度增長快速，因此並沒有很優異的表現，並且在 pr2392 中更是因為記憶體儲存空間不足而無法演算。

兩演算模式運算耗用時間比例比較

平行化 SDAS 的 atomic 及 partition 二模式中，前述兩主要平行運算程式的演算耗用時間在一代次運算時間中所佔的比率列於表 4.6，表中展示這兩種程式求解不同例題的演算時間比例。本測試設定螞蟻數為城市數量，求解 100 代次為停止條件。表中第一欄是測試例題名稱；第二欄列出兩種求解模式；第三欄是平行演算下整體平均每一代次所耗費的總時間；第四欄是建構途程解 kernel 演算時間比；第五欄是更新費洛蒙矩陣 kernel 演算時間比；第六欄是 CPU 演算部分耗用時間比。

表4.6.　兩主要演算模式運算耗時比例

例題名稱	求解模式	總演算時間 (ms)	建構途程解 kernel 演算時間比	更新費洛蒙矩陣 kernel 演算時間比	CPU 演算時間比
att48	SDAS atomic	1.88	97.02%	1.55%	1.43%
	SDAS partition	1.484	94.39%	2.96%	2.65%
krocC100	SDAS atomic	5.22	67.01%	0.64%	32.35%
	SDAS partition	6.3	68.14%	2.16%	29.7%
pcb442	SDAS atomic	91.8	73.04%	0.07%	26.89%
	SDAS partition	116.6	66.62%	12.95%	20.44%
d657	SDAS atomic	245.6	81.76%	0.04%	18.2%
	SDAS partition	302.8	64.46%	17.93%	19.12%
rat783	SDAS atomic	563.5	88.96%	0.02%	11.01%
	SDAS partition	509.3	72.3%	18.8%	8.92%
pr1002	SDAS atomic	901.4	88.6%	0.02%	11.38%
	SDAS partition	968.95	68.132%	19.12%	12.74%
pr2392	SDAS atomic	5457.5	89.5%	<0.01%	10.5%
	SDAS partition	N/A	N/A	N/A	N/A

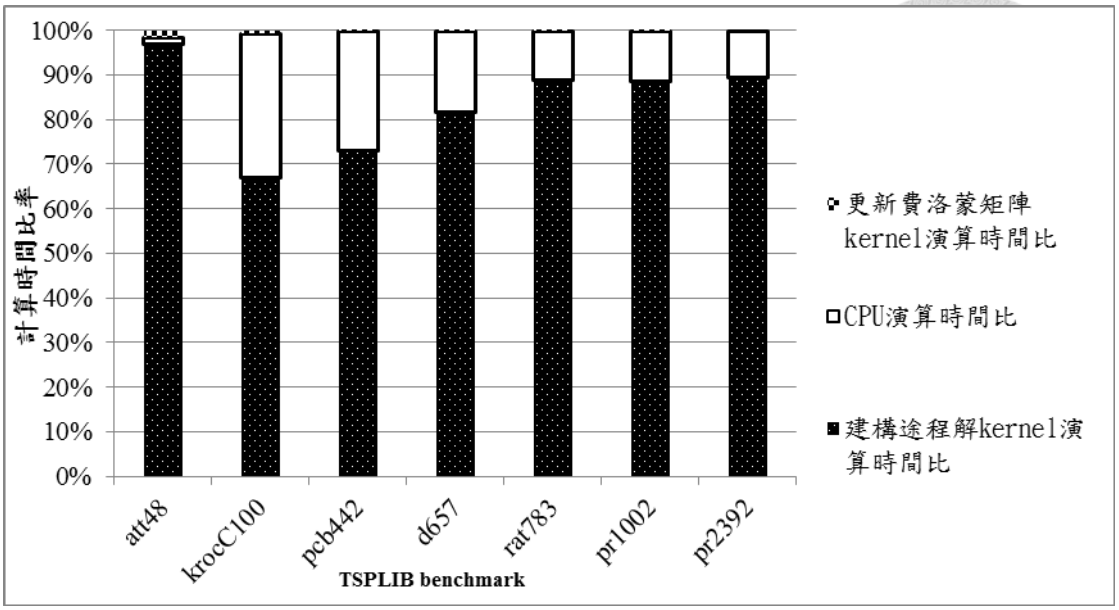


圖 4.5　SDAS atomic 模式各階段演算時間比例

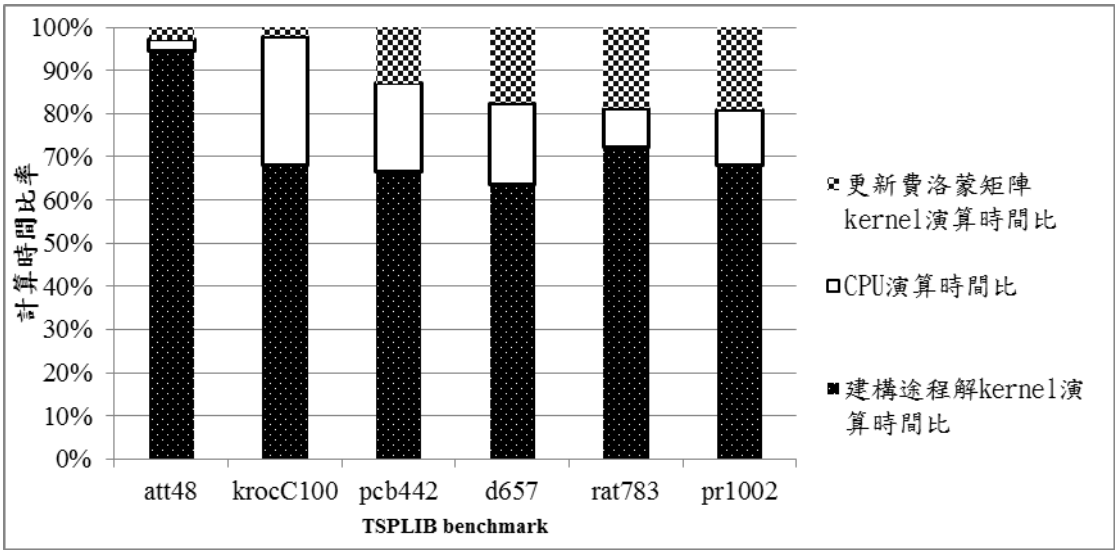


圖 4.6　SDAS partition 模式各階段演算時間比例

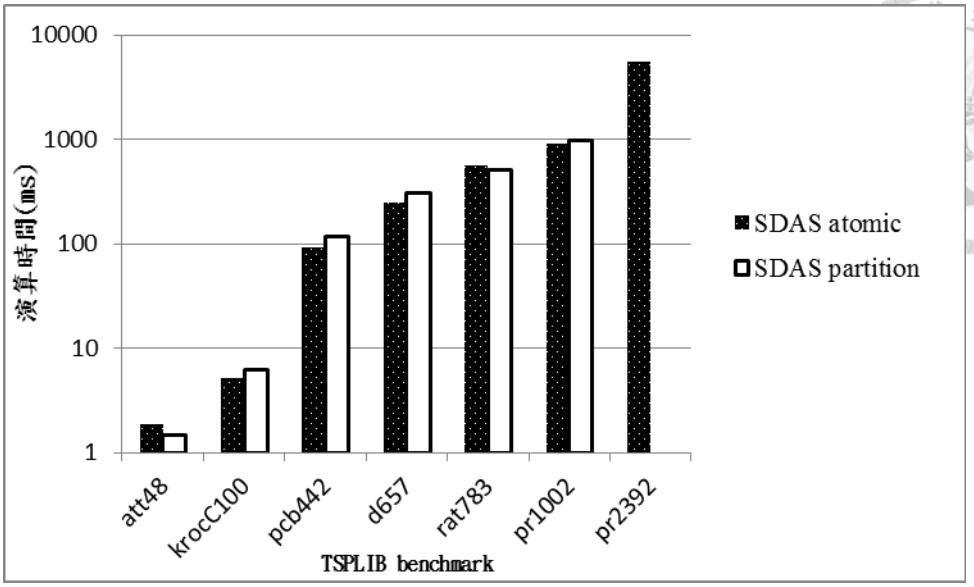


圖 4.7　平行化 SDAS 不同模式下整體演算時間比較

整體上 SDAS atomic 模式的演算速度普遍優於 SDAS partition 模式，在小維度例題上 SDAS atomic 模式因為啟用了 atomic 的運算機制的影響演算效能較 SDAS partition 模式為劣。值得注意的是例題 rat783 SDAS partition 模式的演算速度反超了 SDAS atomic 一點點，本研究認為 SDAS atomic 模式的途程解建構 kernel 的演算速度會隨著問題的維度持續減緩，原因是啟用了 atomic 的運算機制所致，同時 SDAS partition 模式的費洛蒙值更新 kernel 的演算速度亦會隨著問題的維度持續減緩，原因是城市連結標誌矩陣內儲存的資料大幅增加所致。而在約問題維度到達 700 前 SDAS atomic 模式隨著維度大小增加所減緩的演算速度幅度大於 SDAS partition 模式所減緩的幅度，而在之後因 SDAS partition 模式減緩的幅度成等比成長的緣故 SDAS atomic 模式的演算效能又領先回來，因此呈現了例題 rat783 所展示的數據。此現象將於下一小節有進一步的探討。

4.2.3. 整體平行化加速結果比較

本測試比較本研究研擬的 SDAS atomic 及 partition 演算策略，以及文獻上 MMAS (Delévacq, Delisle, Gravel, & Krajecki, 2013)採用 task-based 平行化解建構模式（內部是序列化更新費洛蒙矩陣）求解標竿問題整體效能，重點在統計求解每個代次下的總耗用時間平均及相較於序列模式的 speed up。表 4.7 展示這三種測試而得的演算耗用時間與換算的 speed up。本測試使用螞蟻數為城市數，求解 100 代次為停止條件。表中第一欄是測試例題名稱；第二欄為三種模式下的演算策略；第三欄是平行演算平均每一代次所耗費的時間；第四欄是序列化演算每一代次平均所耗費的時間；第五欄是平行模式相較於序列模式的 speed up。

表4.7.　演算法整體運算耗時及加速成效比較

例題名稱	求解模式	平行演算耗時 (ms)	序列演算耗時 (ms)	Speed up
ei51	SDAS atomic	1,972	6.145	3.12×
	SDAS partition	1.618	6.635	4.1×
	MMAS	N/A	N/A	6.84×
krocA100	SDAS atomic	4.736	24.25	5.12×
	SDAS partition	5.624	25.15	4.47×
	MMAS	N/A	N/A	4×
d198	SDAS atomic	16.34	172	10.53×
	SDAS partition	23.52	185.3	7.87×
	MMAS	N/A	N/A	11.13×
lin318	SDAS atomic	42	691.85	16.47×
	SDAS partition	59.9	767.53	12.81×
	MMAS	N/A	N/A	11.03×
rat783	SDAS atomic	572.3	10187	17.8×
	SDAS partition	511.7	11266.28	22.02×
	MMAS	N/A	N/A	15.58×
f1577	SDAS atomic	2263	82013.4	36.24×
	SDAS partition	3058	91558.75	29.93×
	MMAS	N/A	N/A	19.47×
d2103	SDAS atomic	4267	194711	45.63×
	SDAS partition	N/A	N/A	N/A
	MMAS	N/A	N/A	17.64×

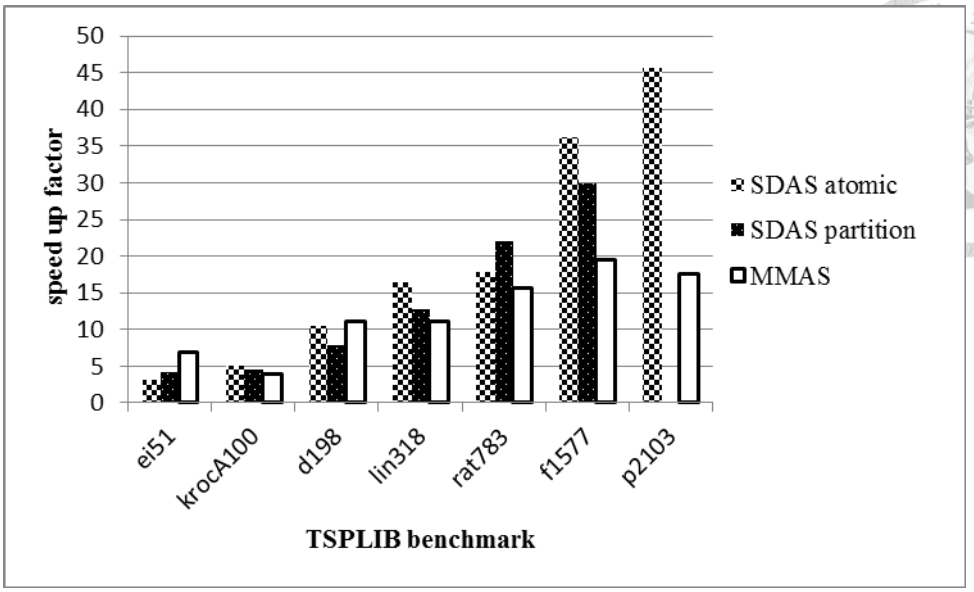


圖 4.8　平行化 SDAS 與 MMAS 的 speed up 比較

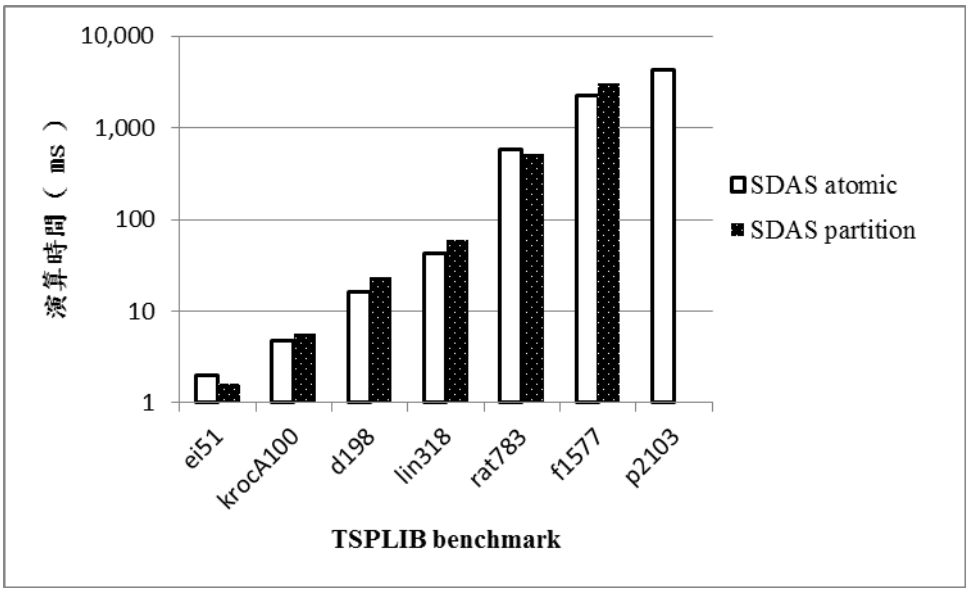


圖 4.9　平行化 SDAS 與 MMAS 的演算時間比較

在圖 4.8 與圖 4.9 對照下，SDAS partition 模式的 Speed up 較 SDAS atomic 模式突出，但演算耗時卻劣於 SDAS atomic 模式，由此本研究認為 SDAS partition 模式較能夠發揮 GPU 的效能，但由於資料量龐大的緣故，SDAS atomic 模式儘管啟用了 atomic function 機制，但相對於 SDAS partition 模式的 *m* 個城市連結標誌矩陣，SDAS atomic 模式的標誌矩陣的數量只有一個，造成兩求解模式的效能有決定性的差異。

4.2.4. atomic 及 partition 兩種資料存取模式的結果比較

由上節展示的演算時間比例及 speed up 資料顯示，求解不同維度的標竿問題，耗時及 speed up 會有差異。表 4.7 例題 rat782 使用 atomic 演算模式的 speed up 一度遜於 partition 模式。派遣蟻次和總演算代次為求與其他研究在比較上的公平，調整參數設定維持一致。

比較效能時本研究除了區別問題維度外，也嘗試了解設定的螞蟻數及總演算代次是否會影響最終的 speed up。本試驗使用例題 rat783 設定不同的螞蟻數與求解代次上限進行測試，表 4.8 展示試驗結果。表中上半部分變動求解代次上限，螞蟻數維持 30。下半部變動螞蟻數，求解代次數上限固定為 100 次。表中第一、二欄分別是螞蟻數及求解代次上限；第三、四、五欄分別是 atomic 模式的序列化演算時間、平行化演算時間、以及換算出的 speed up。第六、七、八欄則為 partition 模式的相對資料。

表4.8.　在不同螞蟻數與不同求解代次上限下求解 rat783 的運算時間和加速結果比較

螞蟻數	求解代次數	atomic			partition		
		序列化演算耗時（ms）	平行化演算耗時（ms）	speed up	序列化演算耗時（ms）	平行化演算耗時（ms）	speed up
30	100	50638	8367	6.05×	55341	14760	3.75×
30	200	101358	16676	6.08×	110796	29611	3.74×
30	300	151958	25006	6.08×	166117	42910	3.87×
30	400	200600	33378	6×	223410	57465	3.89×
30	500	253887	41742	6.08×	277163	74167	3.74×
螞蟻數	求解代次	atomic			Partition		
		序列化演算耗時（ms）	平行化演算時間（ms）	speed up	序列化演算耗時（ms）	平行化演算耗時（ms）	speed up
50	100	76580	9867	7.79×	83420	16021	5.21×
100	100	142239	13061	10.89×	153997	18450	8.35×
150	100	204856	15666	13.08×	225393	20954	10.76×
200	100	269456	18942	14.23×	294701	23006	12.81×
250	100	335210	22140	15.14×	365766	25389	14.41×
450	100	590004	35120	16.8×	659444	34986	18.85×
650	100	848393	47851	17.73×	975021	46329	21.05×

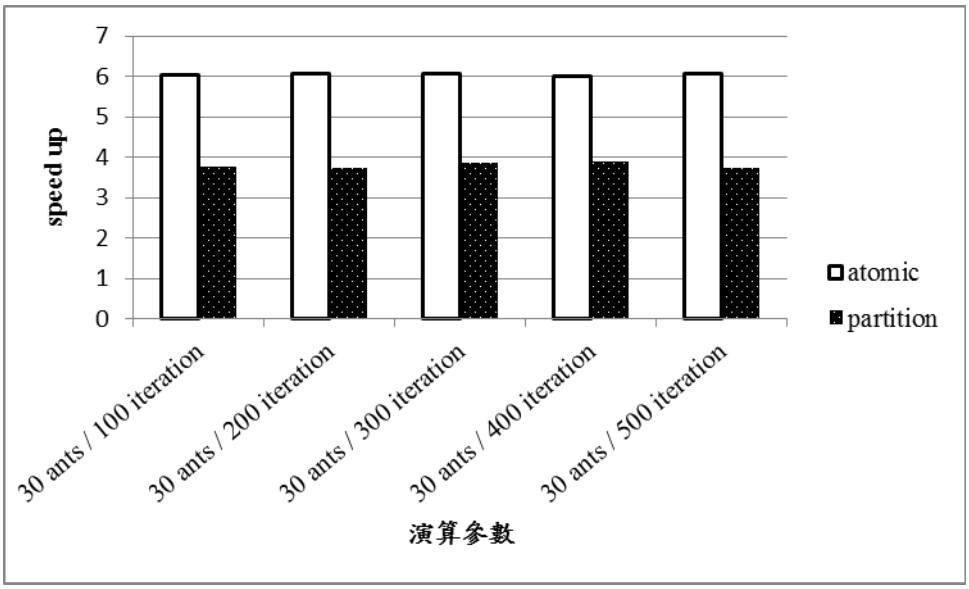


圖 4.10　求解 rat783 下變動求解代次上限的 speed up 比較

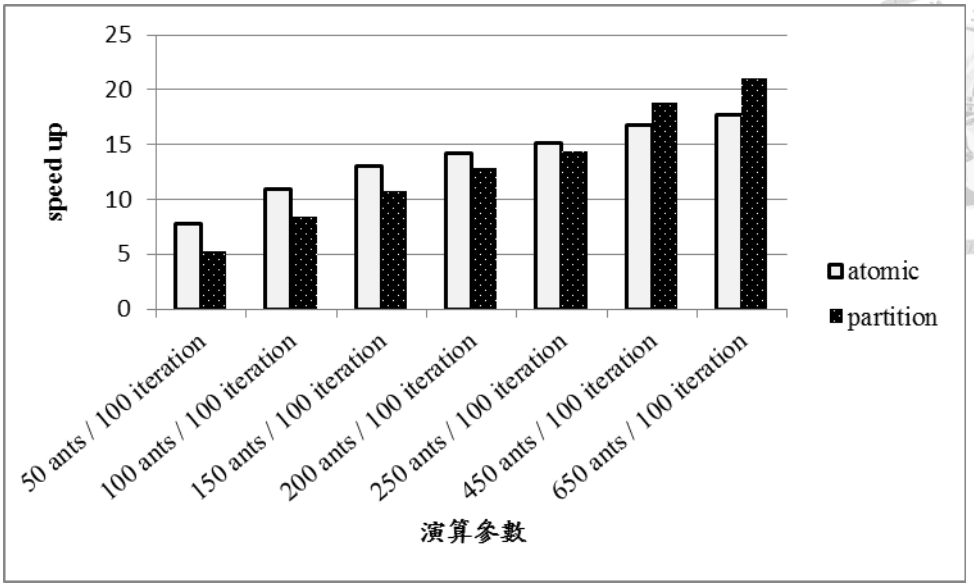


圖 4.11　求解 rat783 下變動螞蟻數的 speed up 比較

圖 4.10 及圖 4.11 可觀察出本研究所提出的兩個平行化策略下，求解代次上限不會影響求解速度，但蟻次越大 SDAS partition 模式及 SDAS atomic 模式兩者的 speed up 皆可增長，在追求演算品質上著實為可作為重要參考的依據。就本研究所展示的成果而言，在加速演算層面上，演算過程中若要增加呼叫目標函數值的次數，則增加螞蟻數會比增加求解代次上限來的有意義。

4.3. 小結

本研究確認平行化版本的 SDAS 保存了原序列化版本的求解品質後，便進行一連串的求解效能分析。

在演算加速結果的比較中，本研究觀察到在大維度問題下 SDAS atomic 模式有顯著的優勢，然而在中偏小的維度下 SDAS partition 模式與 SDAS atomic 模式的效能則相去不遠，而在小維度時 SDAS partition 模式有較亮眼的表現。

在對兩個 kernel 分別做演算耗時與 speed up 比較後，得知 SDAS atomic 模式演算速度上的瓶頸在途程解建構的 kernel 中，相對地 SDAS partition 模式演算速度上的瓶頸則是座落在更新費洛蒙值的 kernel 中。

最後變動螞蟻數維持求解代次上限恆定的實驗中得知，速演算層面上，演算過程中若要增加呼叫目標函數值的次數，則增加螞蟻數會比增加求解代次上限來的有意義。本研究認為求解設定的螞蟻數在大到一定值後 SDAS partition 模式演算速度最終都會超越 SDAS atomic 模式。

第五章　結論與未來研究建議

本研究提示了應用在間接更新 SDAS 費洛蒙值的城市連結標誌矩陣，並使用 CUDA 平行演算程式開發平台平行化 SDAS 演算法，並設計兩種城市連結標誌矩陣儲存方式，提出 SDAS atomic 模式與 SDAS partition 模式的平行化演算法。

5.1. 結論

本研究使用 SDAS atomic 模式與 SDAS partition 模式與其他研究做求解標竿問題的加速效能比較分析後，結果有

1. 本研究所提出的 SDAS 保存了原序列化版本的求解品質。
2. 在大維度問題下 SDAS atomic 模式加速效果較好，在中偏小的維度下 SDAS partition 模式與 SDAS atomic 模式的加速效果差不多，而在小維度時 SDAS partition 模式加速效果較好。
3. SDAS atomic 模式演算速度上的瓶頸在途程解建構的 kernel 中，而 SDAS partition 模式演算速度上的瓶頸則是座落在更新費洛蒙值的 kernel 中。
4. 若要增加呼叫目標函數值的次數，則在平行化蟻拓演算法中增加螞蟻數會比增加求解代次上限來的有意義。
5. SDAS atomic 模式在大型範例下求解速度效能相較 MMAS 與 AS 下依舊穩健。

5.2. 未來研究建議

優加劣減螞蟻系統自 2004 年被提出後，已成功在眾多的組合最佳化問題中大放異彩，本研究所提出幾點建議，供後續研究參考：

1. SDAS atomic 模式在本研究中模式穩健的程度值得在做進一步的試驗分析，是否在更嚴苛的問題下（如更大的維度）SDAS atomic 模式的加速效能便抵達臨界而無法進一步加速？

2. 可嘗試將本研究所提出的演算模式應用於其他的組合最佳化問題上，拓展 SDAS 平行化演算的應用。

3. SDAS partition 模式，有極佳的加速能力，但由於演算時記憶體的不足而無法在高維度下應用，未來可嘗試針對 SDAS partition 模式下的資料結構做進一步的變動，期望達成不需要啟用 atomic 機制的 SDAS partition 模式。

參考文獻

Catala, A., Jaen, J., & Modioli, J. A. (2007, 25-28 Sept. 2007). *Strategies for accelerating ant colony optimization algorithms on graphical processing units.* Paper presented at the Evolutionary Computation, 2007. CEC 2007. IEEE Congress on.

Cecilia, J. M., Garcia, J. M., Ujaldon, M., Nisbet, A., & Amos, M. (2011, 16-20 May 2011). *Parallelization strategies for ant colony optimisation on GPUs.* Paper presented at the Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on.

Crainic, TeodorGabriel, & Toulouse, Michel. (2003). Parallel Strategies for Meta-Heuristics. In F. Glover & G. Kochenberger (Eds.), *Handbook of Metaheuristics* (Vol. 57, pp. 475-513): Springer US.

Delévacq, Audrey, Delisle, Pierre, Gravel, Marc, & Krajecki, Michaël. (2013). Parallel Ant Colony Optimization on Graphics Processing Units. *Journal of Parallel and Distributed Computing, 73*(1), 52-61. doi: http://dx.doi.org/10.1016/j.jpdc.2012.01.003

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 26*(1), 29-41. doi: 10.1109/3477.484436

Hongtao, Bai, Dantong, Ouyang, Ximing, Li, Lili, He, & HaiHong, Yu. (2009, 7-9 Dec. 2009). *MAX-MIN Ant System on GPU with CUDA.* Paper presented at the Innovative Computing, Information and Control (ICICIC), 2009 Fourth International Conference on.

Pedemonte, Martín, Nesmachnow∗, Sergio, & Cancela, Héctor. (2011). A survey on parallel ant colony optimization. *Applied Soft Computing, 11*(71), 5181–5197. doi: 10.1016/j.asoc.2011.05.042

The, x, Van, Luong, Melab, N., & Talbi, E. (2013). GPU Computing for Parallel Local Search Metaheuristic Algorithms. *Computers, IEEE Transactions on, 62*(1), 173-185. doi: 10.1109/TC.2011.206

Wang, Jiening, Dong, Jiankang, & Zhang, Chunfeng. (2009, 11-14 Aug. 2009). *Implementation of Ant Colony Algorithm Based on GPU.* Paper presented at the Computer Graphics, Imaging and Visualization, 2009. CGIV '09. Sixth International Conference on.

Weihang, Zhu, & Curry, J. (2009, 11-14 Oct. 2009). *Parallel ant colony for nonlinear function optimization with graphics hardware acceleration.* Paper presented at the Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on.

Yang, Feng-Cheng, & Chou, Yon-Chun. (2009). Superior/Inferior Segment-Discriminated Ant System for combinatorial optimization problems. *Computers & Industrial Engineering, 57*(2), 475-495. doi: http://dx.doi.org/10.1016/j.cie.2007.12.016

林典翰. (2004). 優加劣減螞蟻擇段系統應用於組合問題. *臺灣大學工業工程學研究所學位論文, 臺灣大學.*

附錄

途程解建構 kernel 原始程式碼

```
1   __global__ void ContructSolutionPathCuda( float *Objective,…argument...){
2   extern __shared__ int sharedMemory[4 * ( 4 + 3 * n )];
3   //SM 內共用變數
4   float* temp, Tabu, Random, solutionResult, WalkingResult; //設定供 share
5   //memory 使用的指標
6   int* ind , tempant;                          //記錄城市編碼
7   //進行 block 內共用資料設定
8   int cityID    = threadIdx.x;
9   int antID = blockIdx.x;
10  //share memory  指標設定
11  Tabu=(float*)&sharedMemory[4];
12  tempant=&sharedMemory[4+ n ];
13  solutionResult=(float*)&sharedMemory[4+ n + n ];
14  ind=&sharedMemory[4+ n + n + n ];
15  temp=(float*)&sharedMemory[2];
16  WalkingResult=(float*)&sharedMemory[3];
17  curandState localState = globalState[cityID]; //亂數產生器
18  curand_init (亂數種子,threadIdx.x, blockIdx.x+1, & localState);
19  //每個 thread  都有自己的種子
20  Tabu[cityID]=1;       //初始化 tabu 值
21  if(threadIdx.x==0)  {//決定起始城市
22     sharedMemory[0]=(int)(curand_uniform( &localState )* n );
23     tempant[0]=sharedMemory[0];
24     ind[0] =sharedMemory[0];
25  }
```

```
__syncthreads();
Tabu[sharedMemory[0]]=0;   //起始城市的 tabu 值設為 0
__syncthreads();
for(int i=1;i<n;i++){//利用轉移機率計算所有城市的隨機輪盤值
                                利用 reduction max 得出選中的城市後繼續迴圈
                                直到所有城市建構完成
   solutionResult[cityID]=  P[ind[0]*n+cityID]/
   (curand_uniform( &localState ))*Tabu[cityID];
   //用 reduction max 來決定所建構的途程
   Ind[cityID] =cityID;
      for (int stride = 1024; stride >= 1; stride=stride/2) {//假設城市數量小於
                                                          1024
         __syncthreads();
         if (cityID < stride &&( cityID+stride)<n){
            if(solutionResult[cityID]<solutionResult[cityID+stride]){
               solutionResult[cityID]=solutionResult[cityID+stride];
               ind[cityID]=ind[cityID+stride];
               }
            }
         }
   __syncthreads();
   if(cityID==0){
      //reduction max  選中的城市 ID 為 ind[0]
      tempant[i]=ind[cityID];
      sharedMemory[0]=ind[0];
      Tabu[ind[0]]=0;
   }
   __syncthreads();
}
ant_family[antID*n+cityID]= tempant[cityID];
//////////////////結束途程解建構///////////////
///////利用較佳較差界限法將螞蟻分群///////
```

```
54  __syncthreads();
55  int cityIDtheNext=(cityID+1)%n;
56  //與 cityID 成對紀錄城市與城市間連結資訊
57  __syncthreads();
58  solutionResult[cityID]= ∅[ tempant[cityID]*n+ tempant[cityIDtheNext]];
    //根據解建構的結果記錄城市之間的距離
59  for (int stride = 1024;stride >= 1; stride=stride/2) {//利用 reduction sum 計算目
                                                          標函數值
60    __syncthreads();
61    if (cityID < stride &&( cityID+stride)<n){
62    solutionResult[cityID]=solutionResult[cityID+stride]+solutionResult[cityID];
63    }
64  }
65  __syncthreads();
66  f(S)=solutionResult[0]; //記錄每隻螞蟻的目標函數值
67  __syncthreads();
68  Objective[antID]= f(S);// 將每隻螞蟻的目標函數值用 global
                                                  Memory 紀錄
69  ind[cityID] =cityID;
70  __syncthreads();
71  //利用 reduction min 找出最佳螞蟻
72  if(antID==0) {
73    for (int stride = 1024;stride >= 1; stride=stride/2){
74      __syncthreads();
75      if (cityID < stride &&(cityID+stride)<m){
76        if(Objective[cityID]>Objective[cityID+stride]) {
77          Objective[cityID]=Objective[cityID+stride];
78          ind[cityID]=ind[cityID+stride];
79        }
80      }
81    }
82  }
```

```
83  __syncthreads();

84  //更新迄今最佳解
85  if( f(S*)>Objective[0]&&antID==0){
86      bestSolution[cityID]=ant_family[ind[0]*n+cityID];
87      __syncthreads();
88      f(S*)=Objective[0];
89  }
90  __syncthreads();
91  更新迄今最佳解城市連結對應的費洛蒙值
92  if(antID==0){
93  R [bestSolution[cityID]*n+bestSolution[(cityID+1)%n]]=
94  R [bestSolution[cityID]*n+bestSolution[(cityID+1)%n]]+1/f(S*);
95  }
96  if(antID==0){
97      f_max =Objective[0];//最佳螞蟻
98  }
99  __syncthreads();
100 Objective[antID]= f(S);
101 __syncthreads();
102 //利用 reduction min 找出最差螞蟻
103 ind[cityID] =cityID;
104 if(antID==0){
105     for (int stride = 1024;stride >= 1; stride=stride/2){
106     __syncthreads();
107         if (cityID < stride &&( cityID+stride)<n){
108             if(Objective[cityID]<Objective[cityID+stride]){
109                 Objective[cityID]=Objective[cityID+stride];
110                 ind[cityID]=ind[cityID+stride];
111             }
112         }
```

```
113     }
114  }
115  __syncthreads();
116  if(antID==0){
117     f_min =Objective[0]; //最差螞蟻
118  }
119  Objective[antID]=  f(S);
120  __syncthreads();
121  ind[cityID] =cityID;
122  //計算當代函數值平均
123  if(antID==0){
124     for (int stride = 1024;stride >= 1; stride=stride/2){
125        __syncthreads();
126           if (cityID < stride &&( cityID+stride)<n){
127              Objective[cityID]=Objective[cityID+stride]+Objective[cityID];
128        }
129     }
130  }
131  if(antID==0){
132  f̄ =Objective[0]/ n ;
133  __syncthreads();
134  ω̄=  f̄ + ω*( f_max - f̄ );     //管制上限
135  __syncthreads();
136  ω̲=  f̄ - ω*( f̄ - f_min );      //管制下限
137  __syncthreads();
138  if(cityID==0){
139     temp[0]=curand_uniform( &localState );
140  }
141  ///////標誌城市連結標誌矩陣
142  if(使用 patition 方法){
143     if( f(S) > ω̄ ){
```

```
144        if(temp[0] > θ^B ){    //標誌劣段
145            T_partition [antID* n * n +
               tempant[cityID]* n +tempant[cityIDtheNext]]= -1;
146            T_partition [antID* n * n +
               tempant[cityIDtheNext]* n +tempant[cityID]]= -1;
147        }
148    }
149    else if( f(S) < ω ){
150        if(temp[0] > θ^A ){     //標誌優段
151            T_partition [antID* n * n +
               tempant[cityID]* n +tempant[cityIDtheNext]]=3;
152            T_partition [antID* n * n +
               tempant[cityIDtheNext]* n +tempant[cityID]]=3;
153        }
154    }
155 __syncthreads();
156 }
157 else if(使用 atomic 方法){
158    if( f(S) > ω̄ ){
159        if(temp[0] > θ^B ){       //標誌劣段
160            atomicAdd(&T_atomic [tempant[geneIDtheNext]*
               n +tempant[geneID]],-1);
161            atomicAdd(&T_atomic [tempant[geneID]*
               n +tempant[geneIDtheNext]],-1);
162        }
163    }
164    else if( f(S) < ω ){
165        if(temp[0] > θ^A ){        //標誌優段
166            atomicAdd(&T_atomic [tempant[geneIDtheNext]*
               n +tempant[geneID]],3);
```

```
167            atomicAdd(&T_atomic [tempant[geneID]*
                n +tempant[geneIDtheNext]],3);
168         }
169      }
170   }
171   __syncthreads();
}
```

更新費洛蒙值 kernel 原始程式碼

```
1     __global__ void RefreshTpheromoneParallel(…argument…){
2     int pheromoneID=blockIdx.x*1024+threadIdx.x;   //每一個費洛蒙值都由不同
                                                     //執行緒進行更新
3     __syncthreads();
4     if(pheromoneID>= n^2 ){
5     }
6     else if(使用 partition 方法){
7        for(int i=0;i< m−1 ;i++)     {   //所有不同標誌矩陣相同位置上的標誌相加
8           T_partition [ pheromoneID+(i+1)* n^2 ]=
            T_partition [ pheromoneID+(i+1)* n^2 ]+
            T_partition [i* n^2 +pheromoneID];
9        }
10       for(int j=0;j< m−1 ;j++){         //部分標誌矩陣初始化
11       T_partition [j* n^2   +pheromoneID]=1;
12       }
13       if( T_partition [( m−1 )* n^2 +pheromoneID]> m ){   //優段更新
14          R [pheromoneID]=   R [pheromoneID]+1/ f(S^*) ;
15       }
16       if( T_partition [( m−1 )* n^2 +pheromoneID]<  m ){   //劣段更新
17          R [pheromoneID]=   R [pheromoneID]*0.85;
```

```
18      }
19      R[pheromoneID]= R[pheromoneID]*0.9+0.00000001;
20      //全域蒸發
21      T_partition[(m−1)*n^2 +pheromoneID]=1;
22  }
23  else if(使用 atomic 方法){
24      if(T_atomic[pheromoneID]>m){      //優段更新
25        R[pheromoneID]=R[pheromoneID]+1/f(S*);
26      }
27      if(T_atomic[pheromoneID]<m){       //劣段更新
28         R[pheromoneID]= R[pheromoneID]*0.85;
29      }
30      R[pheromoneID]= R[pheromoneID]*0.9+0.00000001;     //全域蒸發

31      T_atomic[pheromoneID]=1;          //標誌矩陣初始化
32  }
33  __syncthreads();
}
```